



**Universidad
Rey Juan Carlos**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO EN INGENIERÍA INFORMÁTICA E INGENIERÍA DE
COMPUTADORES

Curso Académico 2020/2021

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN EDUCATIVA PARA ENSEÑAR
PROGRAMACIÓN CON EL USO DE MAKEY-MAKEY.
PRIMARY CODE V.3.

Autor: José Ignacio Díaz Errejón

Directores: Raquel Hijón Neira

Índice de contenidos

1.	Resumen	4
2.	Introducción	5
3.	Objetivos	7
3.1.	Descripción del problema	7
3.2.	Metodología empleada	9
4.	Mejoras de usabilidad y experiencia del usuario	10
5.	Nueva funcionalidad: Ficheros	13
5.1.	¿Qué es un fichero?	16
5.2.	¿Qué es un fichero de texto?	17
5.3.	Escribir en un fichero de texto 1	18
5.4.	Escribir en un fichero de texto 2	20
5.5.	Leer de un fichero de texto 1	22
5.6.	Leer de un fichero de texto 2	24
5.7.	¿Qué es un fichero binario?	26
5.8.	Escribir en un fichero binario	28
5.9.	Leer de un fichero binario	31
5.10.	Leer de un fichero binario 2	33
6.	Nueva funcionalidad: Funciones	35
6.1.	Funciones: ¿qué es una función?	35
6.2.	Funciones: parámetros y valor devuelto	37
6.3.	Funciones: parámetros sin valor devuelto	40
6.4.	Funciones: valor devuelto sin parámetros	43
6.5.	Funciones: sin parámetros ni valor devuelto	46
6.6.	Funciones: una función más completa	48
7.	Nueva funcionalidad: Recursividad	52
7.1.	Funciones recursivas	52
7.2.	Recursividad lineal: función sumar	54
7.3.	Recursividad lineal: función factorial	57
7.4.	Recursividad por la cola: función sumar	60
7.5.	Recursividad por la cola: función factorial	64
7.6.	Recursividad por la cola: función imprimir suma	67
8.	Web de difusión para la aplicación	69
9.	Conclusiones	72
9.1.	Dificultades y limitaciones	75
9.2.	Mejoras y trabajos futuros	76

10. Bibliografía	77
-------------------------------	-----------

1. Resumen

Primary Code es una aplicación creada por alumnos de la Universidad Rey Juan Carlos cuyo propósito es enseñar de manera didáctica a los más jóvenes sobre conceptos de programación en Java.

En su primera versión (V.1 de 2016) se integró la enseñanza de los conceptos más básicos de la programación. En esta versión nos encontramos con pestañas en las que se enseñan conceptos sobre entrada/salida por pantalla y teclado, condicionales (if-else y switch) y bucles (while, for y do-while). También se agregó la opción de selección de usuario (alumno o profesor).

En su segunda versión (V.2 de 2018) se integró la enseñanza de conceptos más avanzados como pueden los arrays unidimensionales y los arrays bidimensionales o matrices, y se trabajó en mejorar algunas pestañas de la primera versión para adaptarlas a conocimientos más actuales e incorporar nuevos ejemplos. También se incorporó algún minijuego para enseñar a los jóvenes el uso de arrays bidimensionales de una forma entretenida y didáctica, y se añadió la opción de usar un usuario “invitado” para que no fuera necesario registrar un nuevo usuario.

En la tercera versión (V.3 de 2020) de la cual trata este el trabajo de fin de grado, se han incorporado conceptos más complejos como son el tratamiento de ficheros, la utilización de funciones y la recursividad, y se han añadido conceptos más actuales en la sección de arrays para ilustrar a los jóvenes sobre el uso de los arrays de objetos.

A lo largo de todas las versiones de la aplicación, se ha primado el uso de la herramienta Makey-Makey, un hardware similar al mando de una videoconsola que permite enviar órdenes al ordenador al que se encuentre conectado. Gracias a esta herramienta, en la aplicación se ha podido asignar a cada tecla de Makey-Makey una fruta o plastilina para que sea una aplicación más visual para los más jóvenes.

2. Introducción

Con el claro avance de la tecnología a lo largo de los años y la presencia de la era digital en la que la sociedad se está sumergiendo, se han abierto muchas puertas en el sector informático y cada año, más y más jóvenes se atreven a realizar estudios relacionados con la informática abarcando desde la ciberseguridad hasta el fenómeno denominado Big Data. Según un estudio del CODDII – Conferencia de Directores y Decanos de Ingeniería Informática de las Universidades Españolas – los alumnos de Ingeniería Informática que encuentran empleo tras finalizar sus estudios suponen un 83% de los casos.

La sociedad se ha dado cuenta que la informática es una ciencia de futuro ya que actualmente muchos sectores laborales se han visto informatizados (medicina, biología, farmacia, etc.), se ha producido un enorme crecimiento en la influencia ejercida por las redes sociales y que actualmente muchas plataformas de entretenimiento online o streaming han obtenido cifras de visualizaciones cercanas a las que obtienen los sistemas informativos de cada país (prensa o televisión). Según un artículo de Unicef publicado en el año 2017, un 71% de las personas de todo el mundo están conectadas de una forma u otra a la tecnología.

Por todo esto, son muchas las organizaciones que están centrando sus esfuerzos e intereses en enseñar a los jóvenes conceptos relacionados con la programación.

Como he comentado previamente, Primary Code es una aplicación creada por alumnos de la Universidad Rey Juan Carlos que tiene como propósito enseñar de manera didáctica a los más jóvenes sobre conceptos de programación básica en Java. Desde su primera versión en 2016 ha sufrido una serie de cambios tanto estéticos como en términos educativos y nos encontramos ahora ante la tercera versión en 2020, en la cual se enseñan conceptos agregados previamente como entrada/salida, condicionales, arrays y bucles, y conceptos nuevos como ficheros, funciones y recursividad.

En la Tabla 1 se representan las funcionalidades existentes previamente y las funcionalidades y modificaciones añadidas en esta versión para que quede más claro los cambios que se han realizado:

Tabla 1. Comparación entre las funcionalidades de las versiones 2 y 3 de Primary Code.

Funcionalidad Primary Code V2	Número de ejercicios	Funcionalidad Primary Code V3	Número de ejercicios
Ejercicios de Entrada / Salida	5	Ejercicios de Entrada / Salida	5
Ejercicios de Condicionales	8	Ejercicios de Condicionales	8
Ejercicios de Bucles	18	Ejercicios de Bucles	18
Ejercicios de Arrays	11	Ejercicios de Arrays	12
Ejercicios de Ficheros	0	Ejercicios de Ficheros	11
Ejercicios de Funciones	0	Ejercicios de Funciones	6
Ejercicios de Recursividad	0	Ejercicios de Recursividad	6

3. Objetivos

3.1. Descripción del problema

El objetivo de este trabajo de fin de grado es mejorar la aplicación Primary Code y agregar nuevos conceptos importantes en cuanto a la programación básica en Java. La programación, tanto en Java como en otros lenguajes, abarca una amplia variedad de conocimientos y conceptos, desde los más simples como la entrada y salida por pantalla y teclado, hasta los más complejos como puede llegar a ser la programación orientada a objetos.

Dado que Primary Code es una aplicación cuyo propósito es enseñar de una forma visual y clara conceptos básicos de programación a los jóvenes, nos hemos centrado en agregar elementos básicos, pero al mismo tiempo más complejos que los que se enseñaban previamente en la aplicación. Por ello, la meta principal ha sido agregar conceptos de funciones, tratamiento de ficheros y recursividad, y tratar de explicar de la forma más visual y clara posible unos conocimientos tan avanzados como pueden llegar a ser los mencionados previamente.

A continuación, se especifican con detalle todos los objetivos a cumplir durante la realización de este Trabajo de Fin de Grado:

- Rediseñar la pantalla del menú de selección de categoría para incorporar nuevos bloques funcionales de funciones, ficheros y recursividad.
- Mantener una estructura similar en los nuevos bloques funcionales de manera que el usuario no perciba versiones diferentes de la aplicación. Para ello utilizar representaciones usadas en versiones anteriores de la aplicación: variables en memoria como frutas en una cesta, arrays como cajas de bolas de billar, etc.
- Explicar de manera visual el concepto de ficheros distinguiendo a su vez entre ficheros de texto y ficheros binarios.
- Explicar de manera visual cómo se produce la lectura o escritura en un fichero.

- Poner ejercicios de lectura y escritura en ficheros de texto y binarios.
- Explicar de manera visual el concepto de funciones con representaciones gráficas de las mismas. Explicar cómo son las llamadas, cómo se pasan parámetros a una función y cómo se devuelven.
- Poner ejemplos de funciones con parámetros, sin parámetros, con valor devuelto y sin valor devuelto, cambiando el tipo de los parámetros recibidos y devueltos (por ejemplo, usando enteros y strings). En caso de que no se devuelva un resultado imprimir por pantalla.
- Explicar qué es la recursividad y diferenciar entre recursividad lineal y recursividad por la cola.
- Poner ejercicios sobre recursividad lineal y sobre recursividad por la cola que no sean demasiado complejos. Empezar con un ejercicio de suma y aumentar el nivel hacia un ejercicio de factorial. En estos ejercicios que se cambie entre devolver un valor o imprimirlo por pantalla.
- Para lo desarrollado anteriormente, desarrollar la misma funcionalidad de forma visual e interactiva tanto para código Java como para pseudocódigo. Para que de esta forma el usuario pueda seleccionar entre ver el código Java o el pseudocódigo correspondiente al ejercicio.
- Hacer que las nuevas pantallas agregadas previamente se puedan leer tanto en español como en inglés.
- Crear una web de difusión de la aplicación para poder llegar a los colegios e institutos fácilmente, donde se explique qué se puede encontrar y aprender con Primary Code V.3. y donde se proporcione un enlace de descarga de la aplicación.

3.2. Metodología empleada

Para el desarrollo de la aplicación se ha usado el entorno de desarrollo de Netbeans y se ha implementado el código en lenguaje Java.

A la hora de añadir nuevas pestañas o ventanas a la aplicación existente se han usado los contenedores o formularios jFrame existentes en el GUI Builder del entorno de desarrollo, ya que facilita la creación de interfaces proporcionando distintos elementos como los jButton, jLabel, etc.

Respecto a las nuevas pestañas añadidas, como se desea mantener el estilo usado en versiones anteriores, los ejemplos siguen manteniendo la estética de ir ejecutando el código presentado instrucción por instrucción para que los usuarios (alumnos) observen lo que ocurre en cada momento de la ejecución, ya sea que se inicializa una variable en memoria, que se guarda un valor en una variable o que se imprime por pantalla.

4. Mejoras de usabilidad y experiencia del usuario

Dado que se han introducido tres conceptos nuevos de la programación en Java, era necesario crear un acceso desde el menú de selección de categorías a la propia categoría. Esto provocaba que el menú quedase bastante cargado manteniendo la estética propuesta en la versión anterior y que el usuario presentase dificultades en la utilización de este.

Para mejorar la experiencia del usuario se ha cambiado el diseño del menú priorizando el orden de las categorías y manteniendo orden y limpieza en este.

Los requisitos para esta pantalla son los siguientes:

- Mantener el menú ordenado numéricamente y por columnas.
- Aportar una imagen o representación gráfica para cada categoría presente en el menú.

El caso de uso que puede realizar un usuario dentro de esta pantalla de menú son los siguientes:

- Acceder a la información de la aplicación.
- Acceder a una categoría.
- Volver al menú de selección de interacción.

Este caso de uso se puede observar en el Diagrama 1.

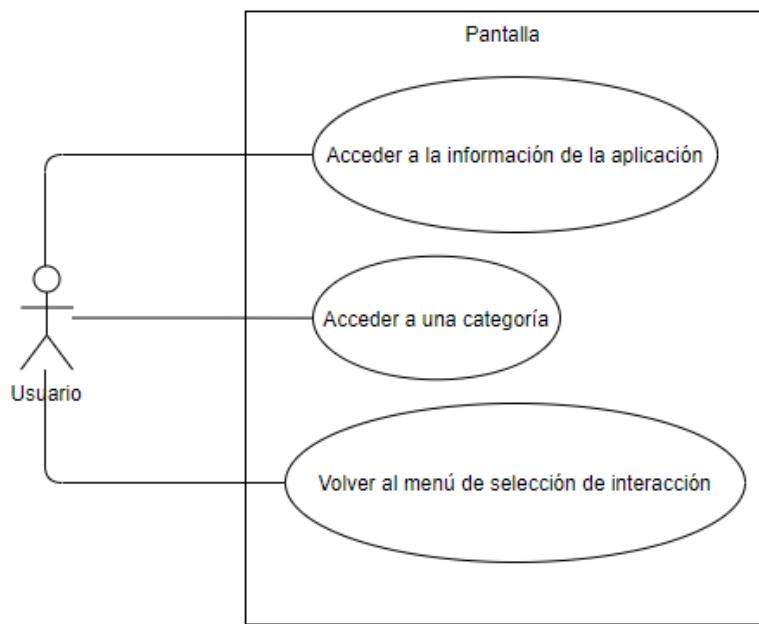


Diagrama 1. Caso de uso: Pantalla de menú de selección de categoría.

Fuente: Elaboración propia.

En la Figura 1, se puede observar cómo quedaba el menú manteniendo la estética antigua. Mientras que en la Figura 2, se puede observar un menú más ordenado y limpio. El menú queda organizado en columnas y filas presentando una ordenación numérica y con varios iconos o imágenes que representan de la mejor manera posible cada categoría.



Figura 1. Pantalla del menú manteniendo el estilo de la versión 2 y añadiendo los nuevos conceptos.

Fuente: elaboración propia



Figura 2. Pantalla del menú aplicando orden y limpieza.

Fuente: elaboración propia.

5. Nueva funcionalidad: Ficheros

Para esta nueva funcionalidad los requisitos iniciales han sido los siguientes:

- Explicar qué es un fichero
- Explicar qué es un fichero de texto y qué es un fichero binario
- Mostrar un ejercicio de lectura de una línea de un fichero de texto
- Mostrar un ejercicio de lectura de varias líneas de un fichero de texto
- Mostrar un ejercicio de escritura de una línea de un fichero de texto
- Mostrar un ejercicio de escritura de varias líneas de un fichero de texto
- Mostrar un ejercicio de lectura de un fichero binario
- Mostrar un ejercicio de escritura de un fichero binario

En cuanto al diseño, como ya existía una versión previa de la aplicación, se ha seguido el diseño existente en cuanto a la arquitectura y a la interfaz. Haciendo hincapié en el diseño de la interfaz:

- Las pantallas de explicación o descripción debían tener un texto explicativo breve y además un ejemplo visual.
- Las pantallas de ejercicios debían tener una sección en la que se muestra el código o pseudocódigo, resaltando en color rojo o con una flecha la sentencia que se está ejecutando en el momento actual y un ejemplo visual y representativo del ejercicio.

En cuanto a los casos de uso que puede realizar un usuario en cada pantalla añadida para esta nueva funcionalidad nos encontramos con dos situaciones:

Caso de uso: Pantalla de explicación o descripción de concepto. Como se puede observar en el Diagrama 2, en este caso el usuario solamente puede realizar cuatro operaciones:

- Volver a la pantalla anterior
- Ir al menú
- Resetear la pantalla

- Ir a la siguiente pantalla

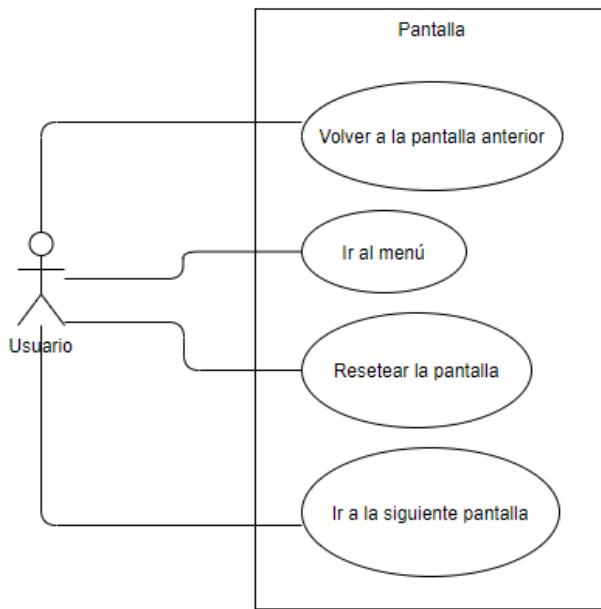


Diagrama 2. Caso de uso: Pantalla de explicación o descripción de concepto.

Fuente: Elaboración propia.

Caso de uso: Pantalla de representación de un ejercicio. Como se puede observar en el Diagrama 3, en este caso el usuario puede realizar las siguientes operaciones:

- Volver a la pantalla anterior
- Ir al menú
- Resetear la pantalla
- Ir a la siguiente página
- Cambiar a código Java
- Cambiar a pseudocódigo
- Introducir un valor mediante un botón
- Ejecutar el siguiente paso

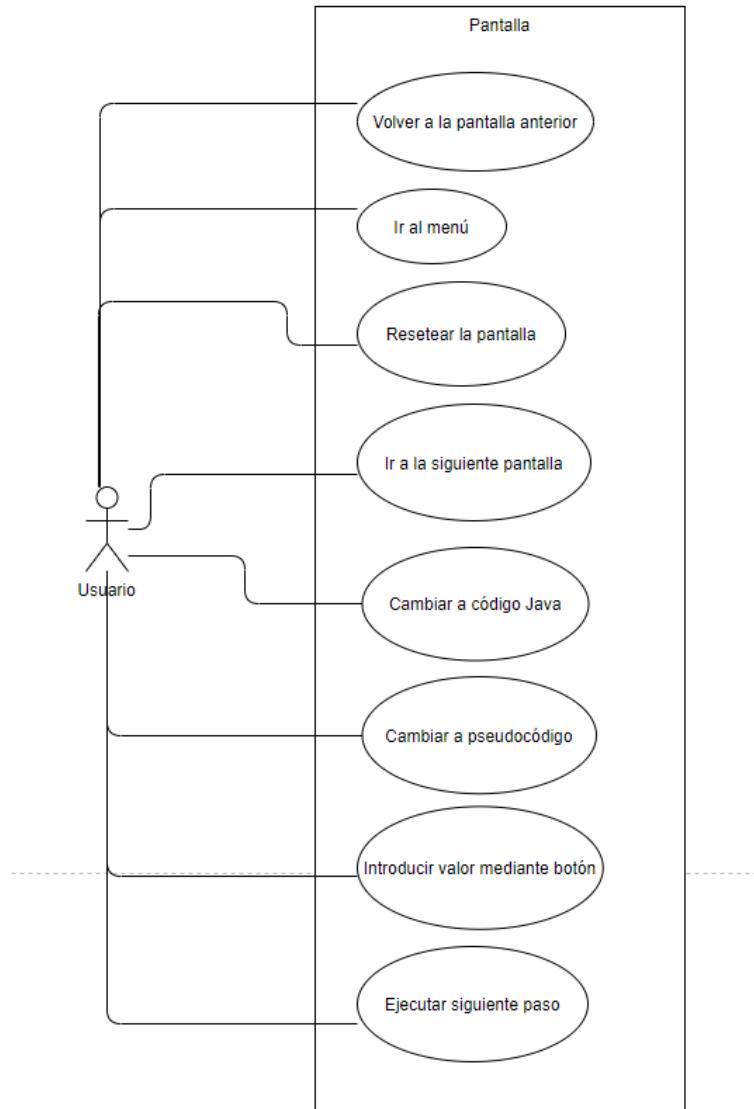


Diagrama 3. Caso de uso: Pantalla de representación de un ejercicio.

Fuente: Elaboración propia.

5.1. ¿Qué es un fichero?

Tras añadir la nueva categoría para ficheros, era necesario tener una pantalla dedicada a explicar de una forma sencilla y comprensible qué es un fichero. En esta pantalla se presenta una breve descripción del término fichero en informática y se hace una especie de representación gráfica de donde se alojan los ficheros dentro de nuestros ordenadores o computadoras y la manera en la que se comunica nuestro programa con un fichero a la hora de usarlo para leer o escribir sobre este.

De esta forma se intenta explicar que, si desde nuestro programa se intenta acceder a un fichero, se abre un flujo que se comunica con la memoria secundaria (disco duro, pendrives, etc.) que es el lugar en el cual están almacenados los ficheros. Esto se puede observar en la Figura 3.

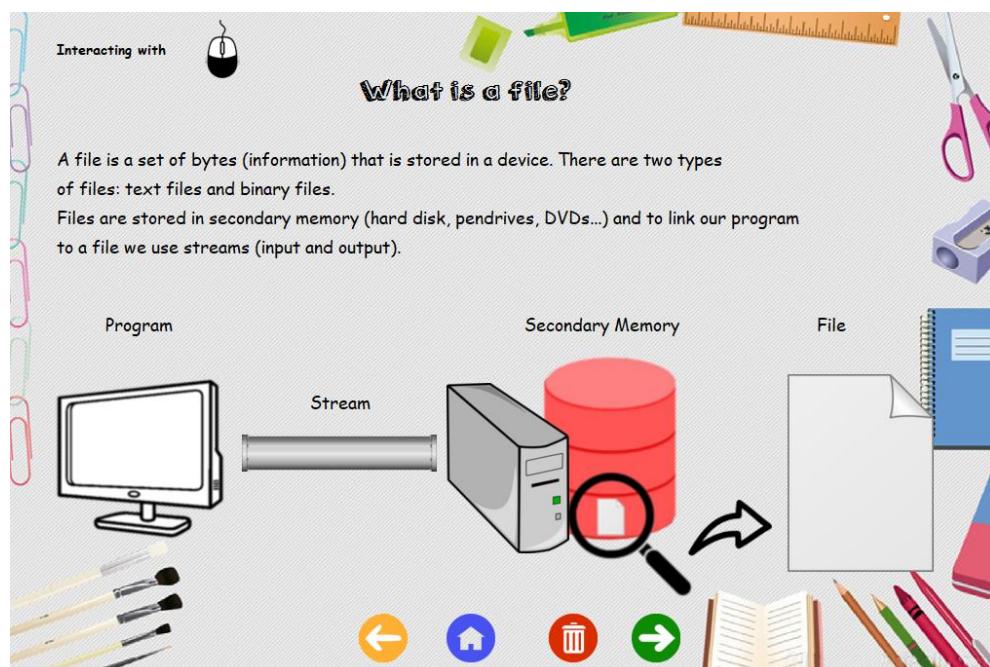


Figura 3. Explicación de ficheros.

Fuente: Elaboración propia.

5.2. ¿Qué es un fichero de texto?

Como hay varios tipos de fichero, se han añadido dos pantallas en las cuales se explica lo que es un fichero de texto y lo que es un fichero binario.

En la primera de ellas la cual podemos ver en la Figura 4 se hace hincapié en explicar que es un fichero de texto y en dar una pequeña descripción de los flujos de entrada y salida que se usan en Java para poder utilizar este tipo de ficheros tanto para leer como para escribir.

- FileReader y BufferedReader para leer de un fichero.
- FileWriter y PrintWriter para escribir en un fichero.

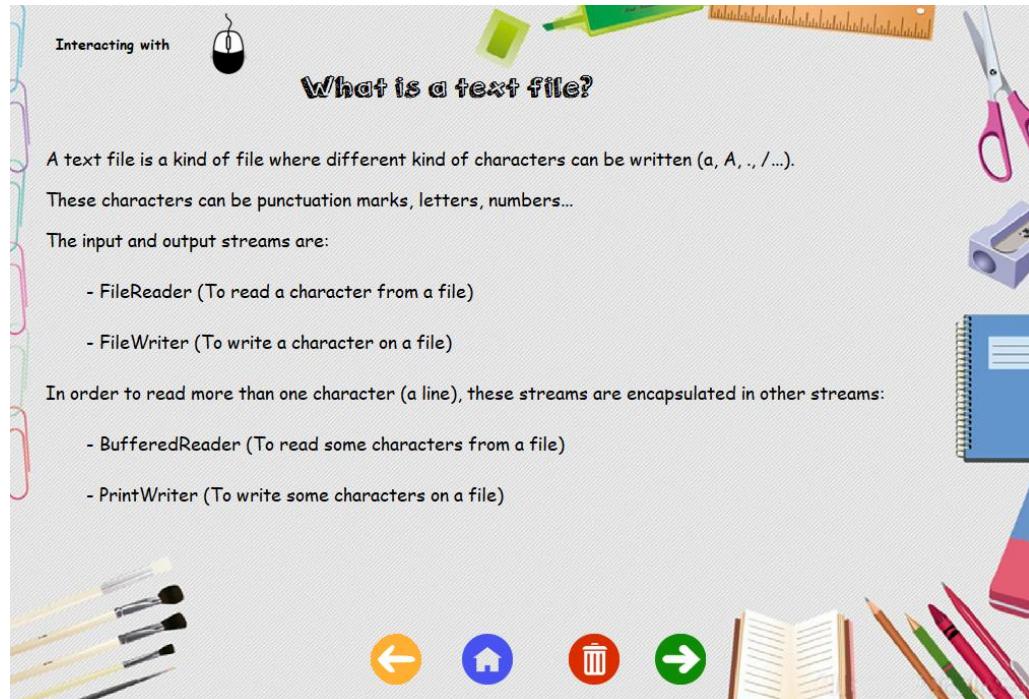


Figura 4. Explicación de ficheros de texto.

Fuente: Elaboración propia.

5.3. Escribir en un fichero de texto 1

En esta pantalla se explica de una manera sencilla cómo se realiza la acción de escribir en un fichero de texto. Para ello se presenta de una manera visual la manera en la que se abre un fichero, se escribe sobre él y se cierra, utilizando para ello los flujos que se han explicado en la pantalla anterior.

En la Figura 5, se puede observar el flujo que se ejecuta a la hora de abrir un fichero. Desde nuestro programa se abre un flujo hacia la memoria secundaria donde se encuentra el archivo y se abre el fichero en cuestión.

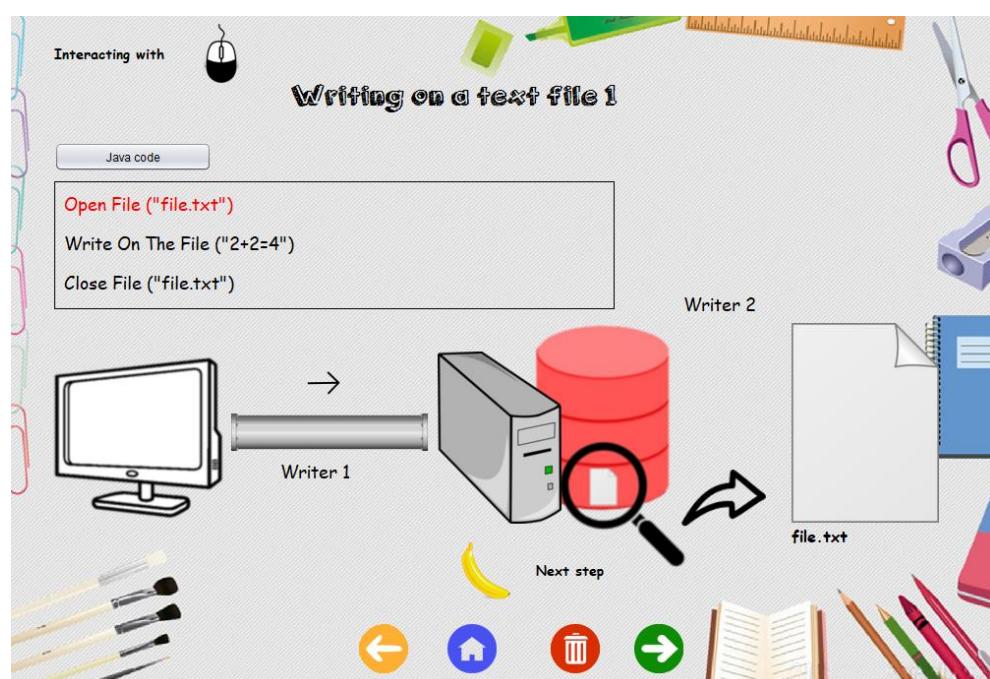


Figura 5. Escritura sobre un fichero de texto 1. Apertura de fichero.

Fuente: Elaboración propia.

En la Figura 6 y Figura 7, se puede comprobar cómo se va pasando el dato desde nuestro programa a través de los flujos hasta llegar al fichero que se quiere modificar y escribir el dato en este.

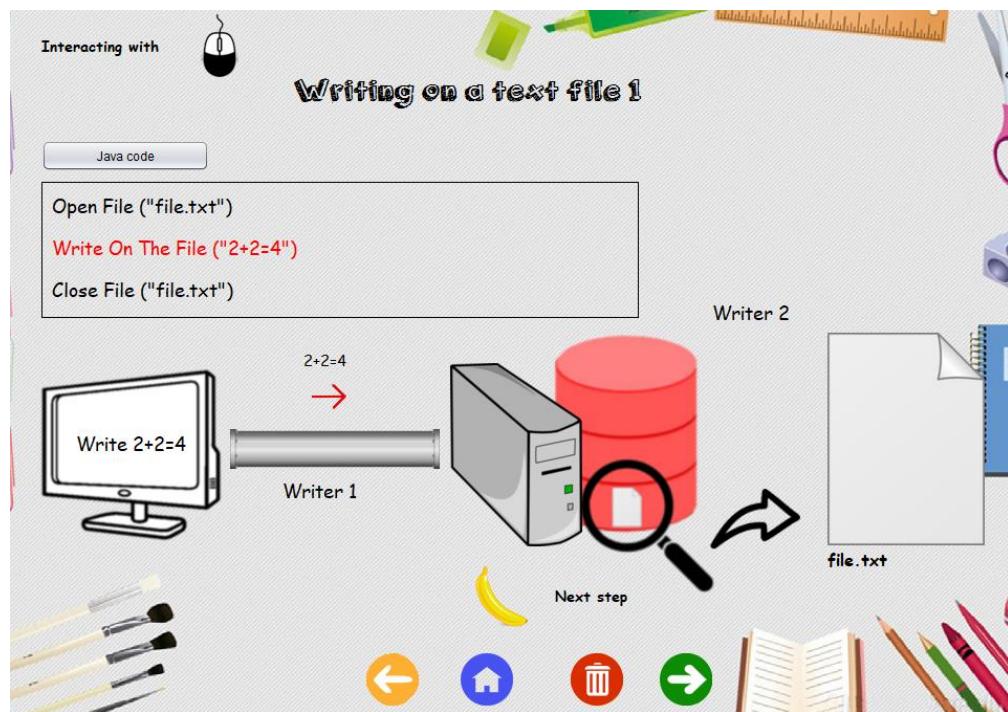


Figura 6. Escritura sobre fichero de texto 1. Escritura.

Fuente: Elaboración propia.

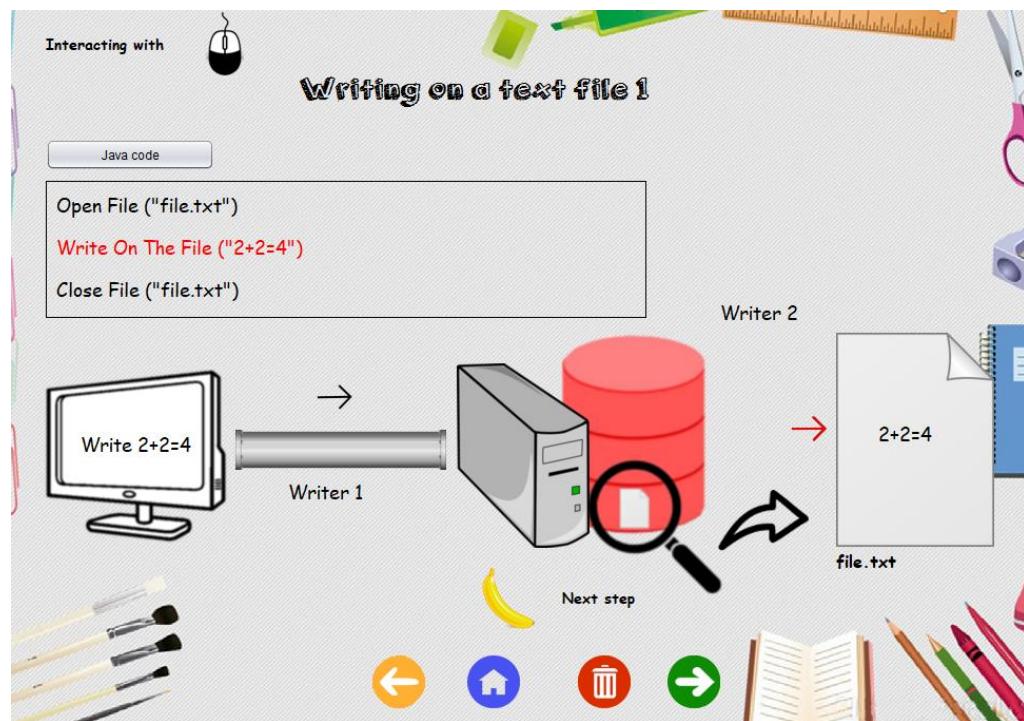


Figura 7. Escritura sobre fichero de texto 1. Dato escrito en fichero.

Fuente: Elaboración propia.

5.4. Escribir en un fichero de texto 2

En esta pantalla se intenta poner un ejemplo de escritura sobre fichero más interactivo, ya que se da al usuario tres opciones para escribir en el fichero (“Hola”, el nombre del usuario y “¿Cómo estás?”). De esta forma el usuario puede elegir que escribir en el fichero en cada momento.

En la Figura 8, se puede observar la pantalla antes de que el usuario elija alguna de las tres posibles opciones a escribir. Para hacer una pantalla mucho más limpia visualmente, se ha simplificado el flujo de manipulación de ficheros, ahora simplemente se presenta el flujo entre la memoria secundaria y el fichero.

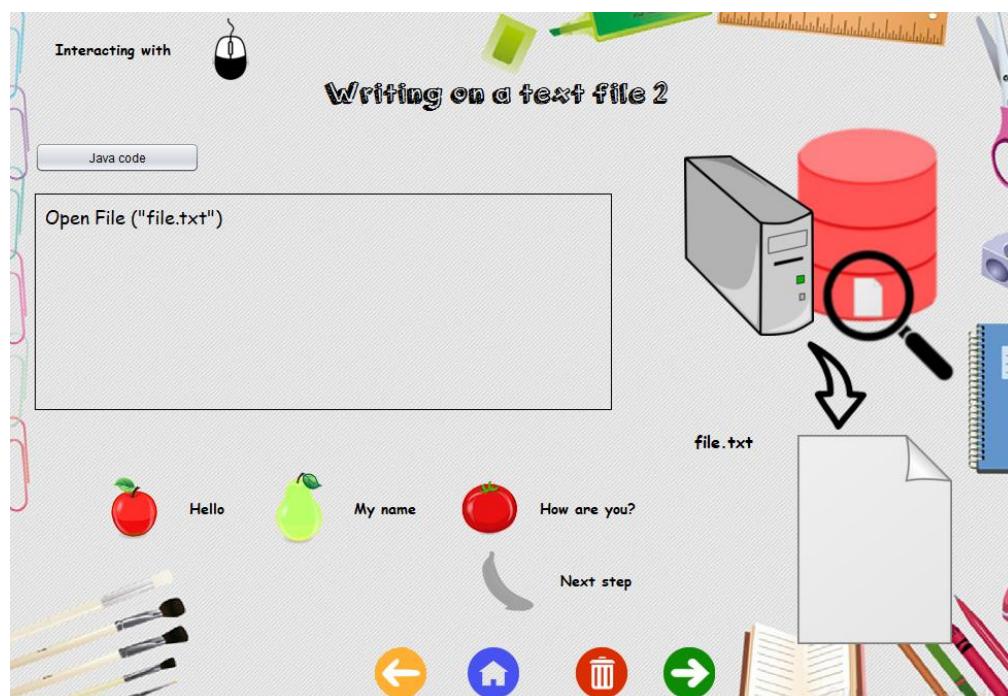


Figura 8. Escribir en un fichero de texto 2. Sin seleccionar ninguna opción.

Fuente: Elaboración propia.

En la Figura 9, se observa que el usuario ha realizado una selección y que dicha selección ha sido impresa o escrita sobre el fichero en cuestión.

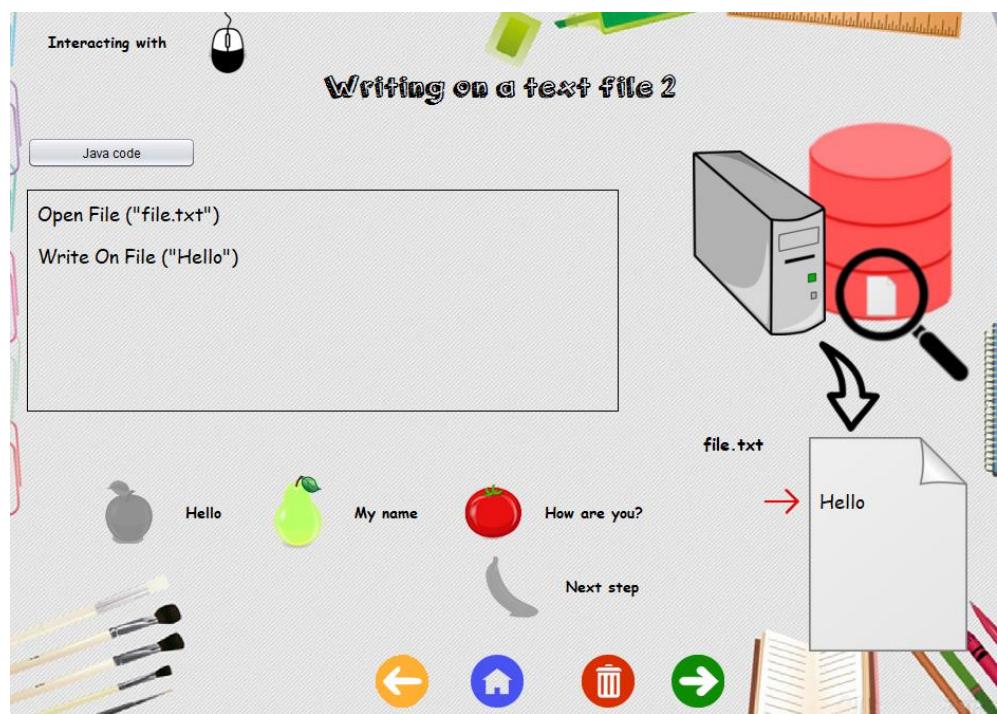


Figura 9. Escribir en un fichero de texto 2. Con opción seleccionada.

Fuente: Elaboración propia.

5.5. Leer de un fichero de texto 1

En esta pantalla se explica de una forma sencilla cómo se realiza la acción para leer de un fichero. Para ello se presenta de una manera visual la manera en la que se abre un fichero, se leen datos de él y se cierra, utilizando para ello los flujos que se han explicado previamente.

En la Figura 10, se representa el flujo que se ejecuta a la hora de leer de un fichero de texto. Si comparamos esta imagen con la “*Figura 5. Nueva pantalla de escritura sobre un fichero de texto 1. Apertura de fichero.*” se puede comprobar que las flechas que indican la dirección del flujo están invertidas para diferenciar si queremos modificar el fichero u obtener valores de él mismo.

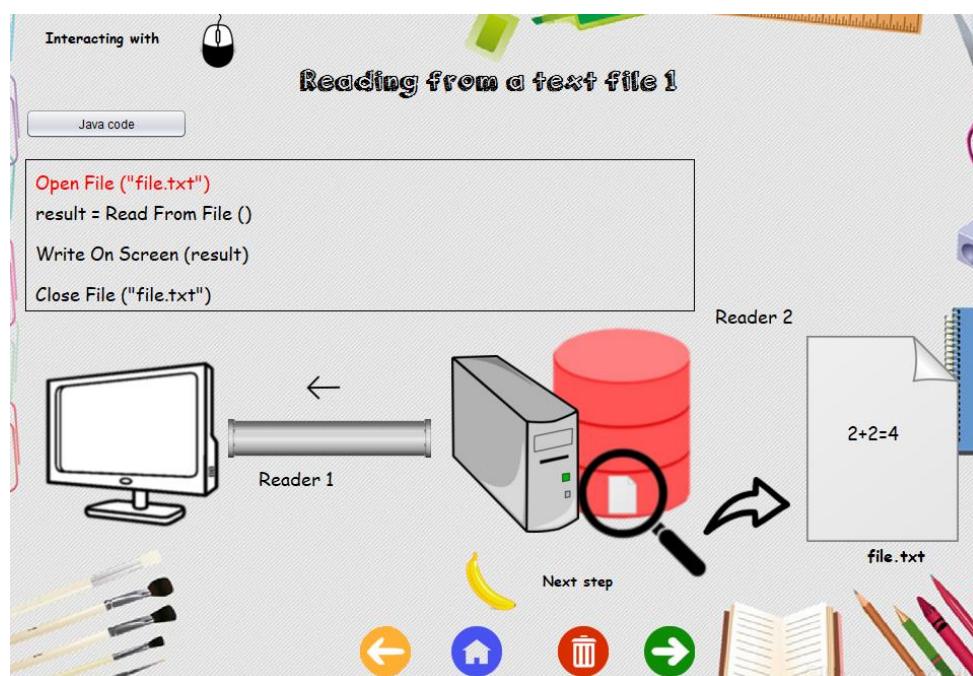


Figura 10. Leer de un fichero de texto 1.

Fuente: Elaboración propia.

En la Figura 11, se observa la obtención de cierto dato del fichero y cómo este dato se guarda en memoria. La representación gráfica de la memoria en esta pantalla se hace mediante una cesta para aclarar que en ella se almacenan datos. De esta forma, la variable en memoria “*result*” almacena el valor del dato obtenido del fichero.

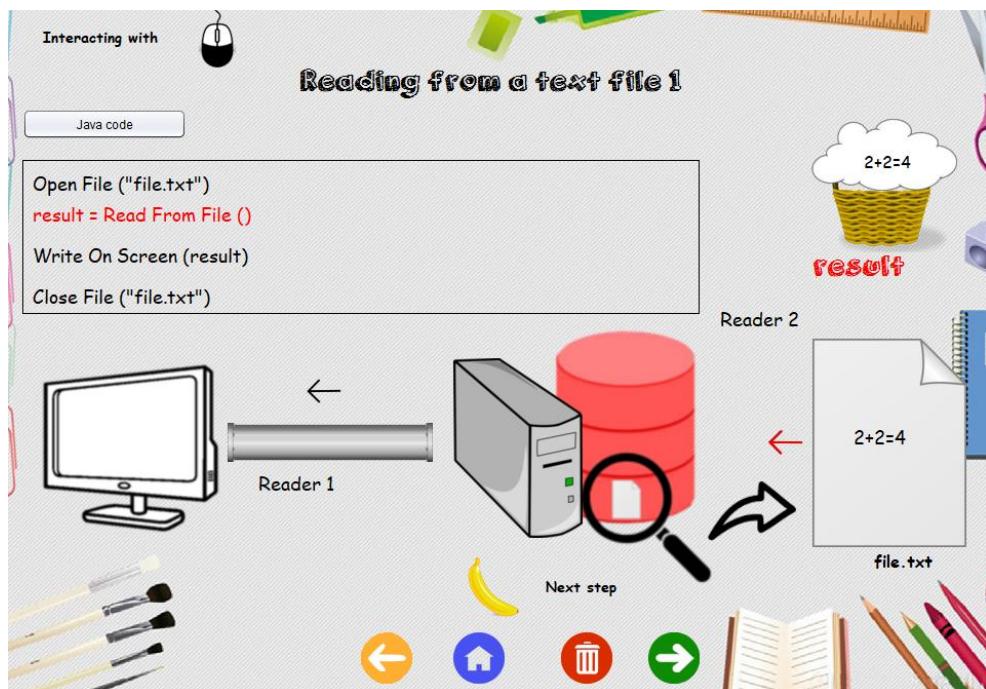


Figura 11. Leer de un fichero de texto 1.

Almacenar dato de fichero en variable en memoria.

Fuente: Elaboración propia.

La variable almacenada en memoria se usa más tarde para mostrar el valor obtenido del fichero imprimiéndolo por pantalla.

5.6. Leer de un fichero de texto 2

En esta pantalla se intenta representar un ejemplo más complejo de lectura de ficheros de texto.

En el fichero tenemos tres datos que queremos leer (“Hola”, “¿Cómo estás?” y el nombre del usuario). Para hacer la lectura se crea un bucle While para recorrer las líneas del fichero hasta que no haya más líneas. Estas líneas se almacenan en la variable “text” en memoria, y por cada nueva linea leída se sobreescribe el valor de la variable.

Finalmente, en cada iteración del bucle se imprime por pantalla el dato leido del fichero.

En la Figura 12, se puede observar que el valor del primer dato obtenido del fichero se almacena en la variable “text” en memoria. Esta representación se hace mediante una cesta.

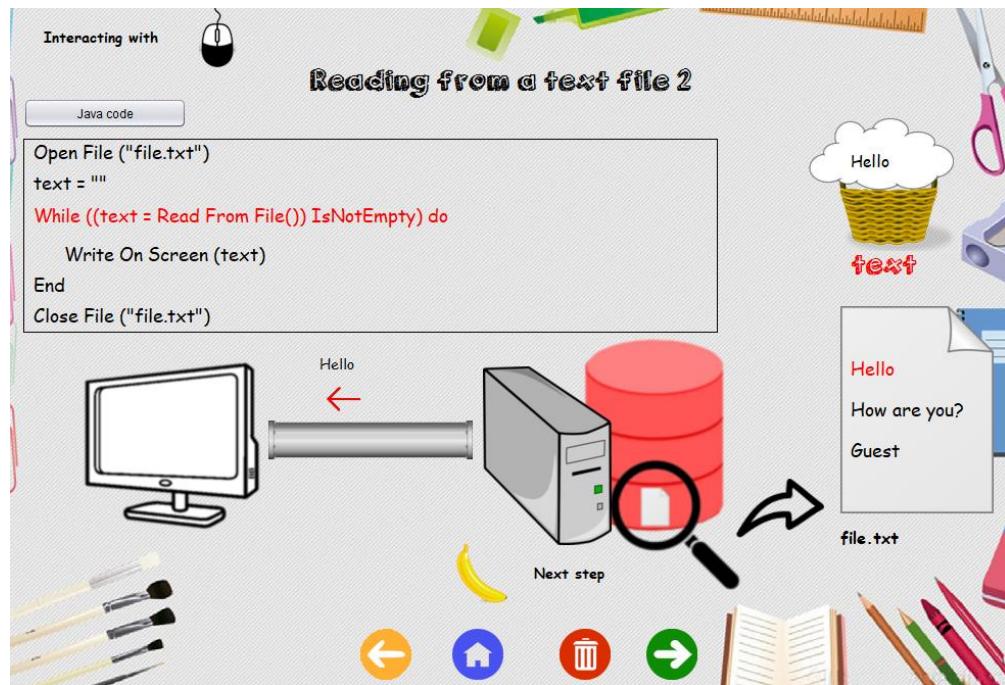


Figura 12. Leer un fichero de texto 2. Almacenar dato en variable en memoria.

Fuente: Elaboración propia.

En la Figura 13, se observa que ya se han leído todos los datos del fichero y se han impreso por pantalla.

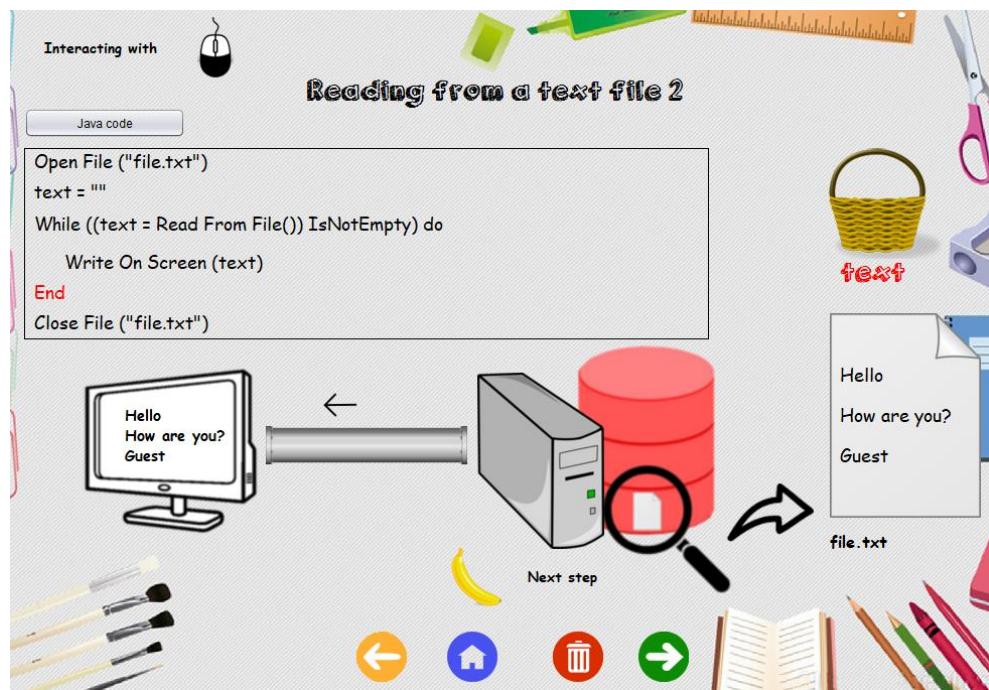


Figura 13. Leer de un fichero de texto 2. Todos los datos mostrados por pantalla.

Fuente: Elaboración propia.

5.7. ¿Qué es un fichero binario?

Al igual que era necesario explicar qué es un fichero y qué es un fichero de texto, también es necesario explicar qué es un fichero binario. Esto se puede observar en la Figura 14. Esta pantalla sirve exactamente para eso, para explicar al usuario en qué consisten los ficheros binarios y los flujos de entrada y salida que se utilizan para escribir y leer sobre estos.

- FileInputStream y ObjectInputStream para leer de fichero.
- FileOutputStream y ObjectOutputStream para escribir en fichero.

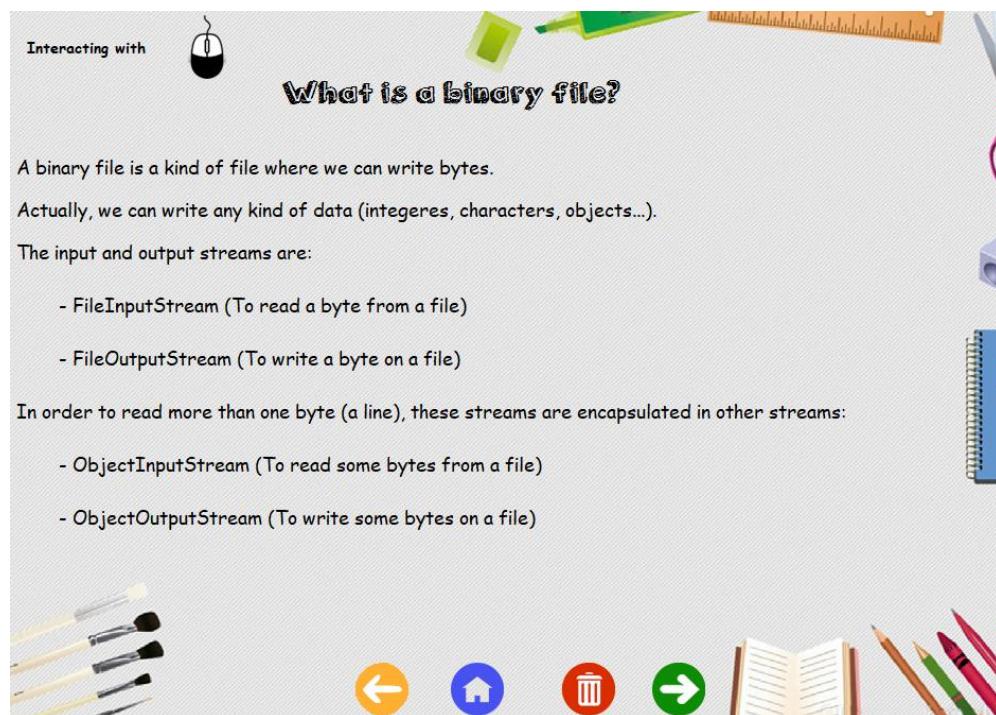


Figura 14. Explicación de fichero binario.

Fuente: Elaboración propia.

En la Figura 15 se explica de una forma simple qué es un objeto (ya que se usan al escribir y leer en ficheros binarios).

Interacting with



What is a binary file?

A binary file is a kind of file where you can save objects.

Data are saved in binary format, with 1's and 0's.

For example:

1001101011110

An object is an element that has properties.

For example:

- A pool ball
- A piece of fruit

Next step



Pool Ball Object

Properties:

- Color -> Red
- Number -> 3

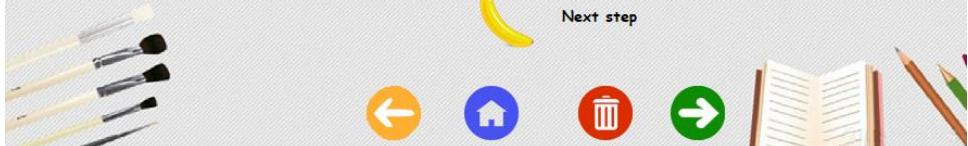


Figura 15. Explicación de fichero binario. Objetos.

Fuente: Elaboración propia.

5.8. Escribir en un fichero binario

En esta pantalla se explica de una forma más o menos avanzada como se realiza una escritura sobre un fichero binario. Para ello, se pone como ejemplo que se tiene un array que contiene tres bolas de billar y se quiere meter cada una de ellas en un fichero binario.

En Figura 16 nos encontramos con una representación gráfica del array, representado como una caja negra que contiene las bolas de billar, una cesta para la variable “*length*” que almacena la longitud del array en memoria y otra cesta para la variable “*i*” que almacena la iteración del bucle en memoria.

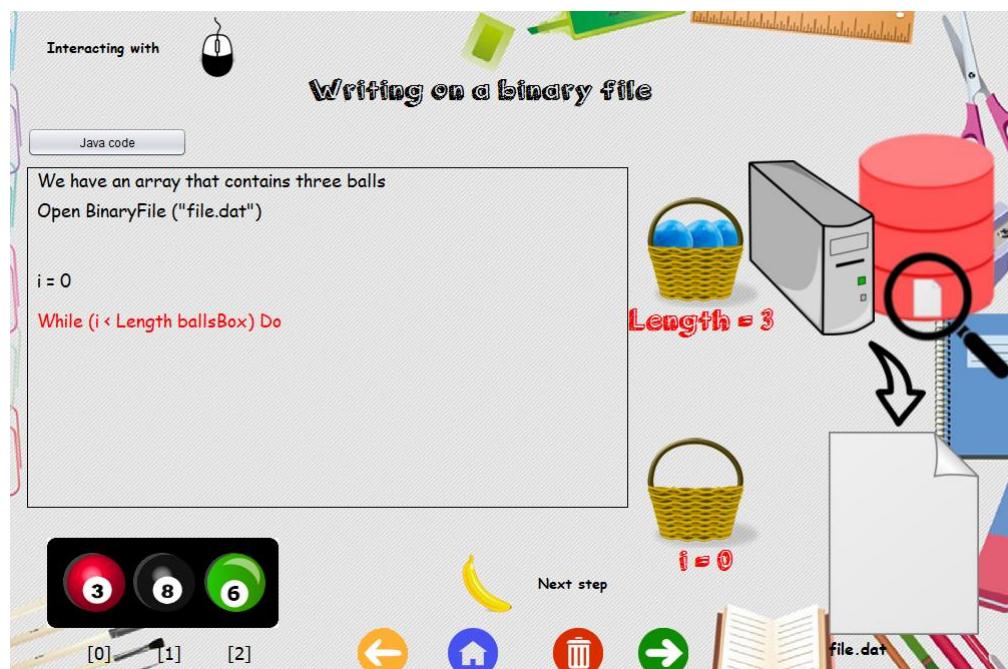


Figura 16. Escribir en un fichero binario. Estado inicial.

Fuente: Elaboración propia.

En la Figura 17, nos encontramos que ya se ha consumido la primera bola de billar y se ha escrito en el fichero. Además se ha aumentado el valor de la iteración.

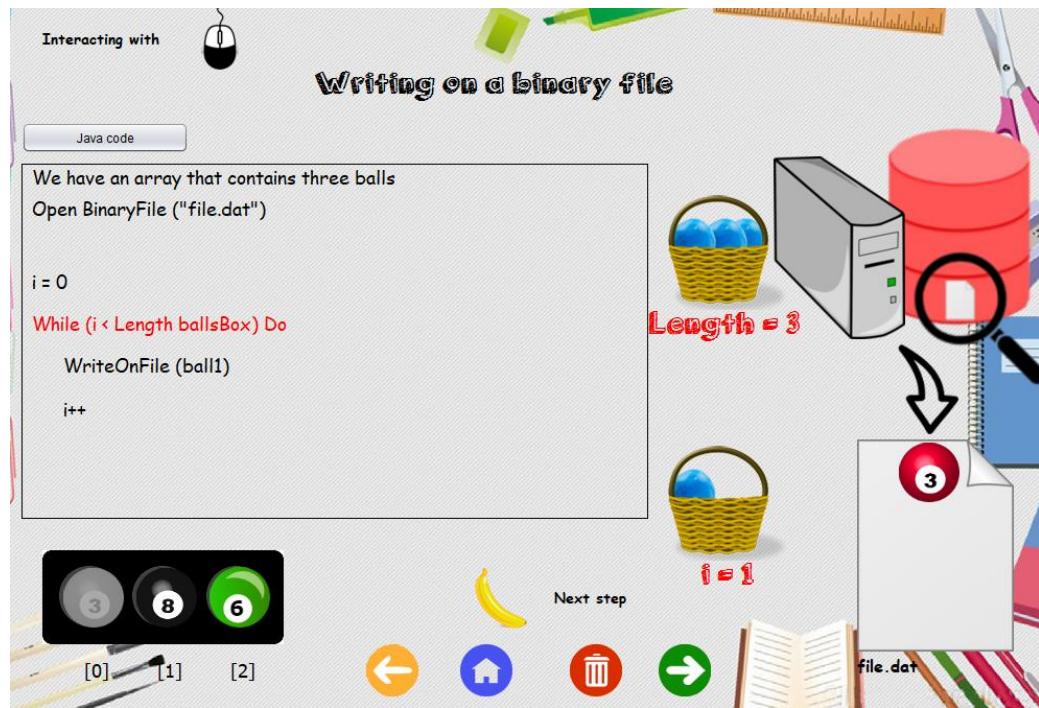


Figura 17. Escribir en un fichero binario. Primer valor escrito.

Fuente: Elaboración propia.

En la Figura 18, se han consumido todas las bolas y se han escrito todos los valores en el fichero. La variable iteradora aumenta para no permitir otra nueva entrada en el bucle.

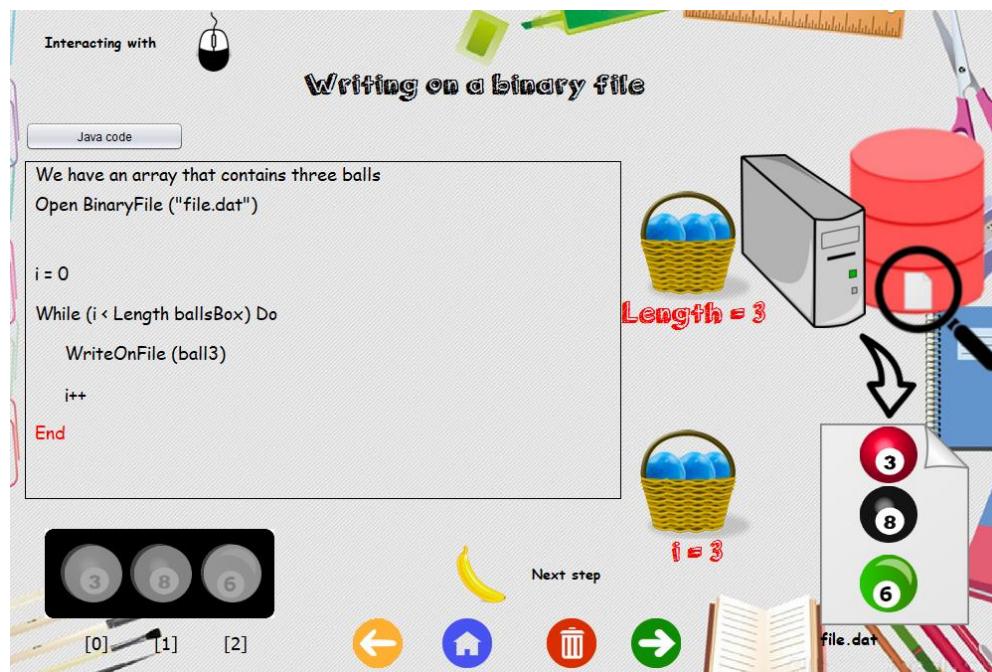


Figura 18. Escribir en un fichero binario. Todos los valores escritos.

Fuente: Elaboración propia.

5.9. Leer de un fichero binario

En esta pantalla, representada en la Figura 19, se utiliza el ejemplo anterior pero a la inversa, es decir, se tiene un fichero que contiene tres bolas de billar y se quiere meter cada una de esas bolas en un array.

Las representaciones gráficas son las mismas que en el ejemplo anterior exceptuando la variable “*length*” que en este caso ha desaparecido y ha aparecido la variable “*ball*” que almacenara cada bola leída del fichero.

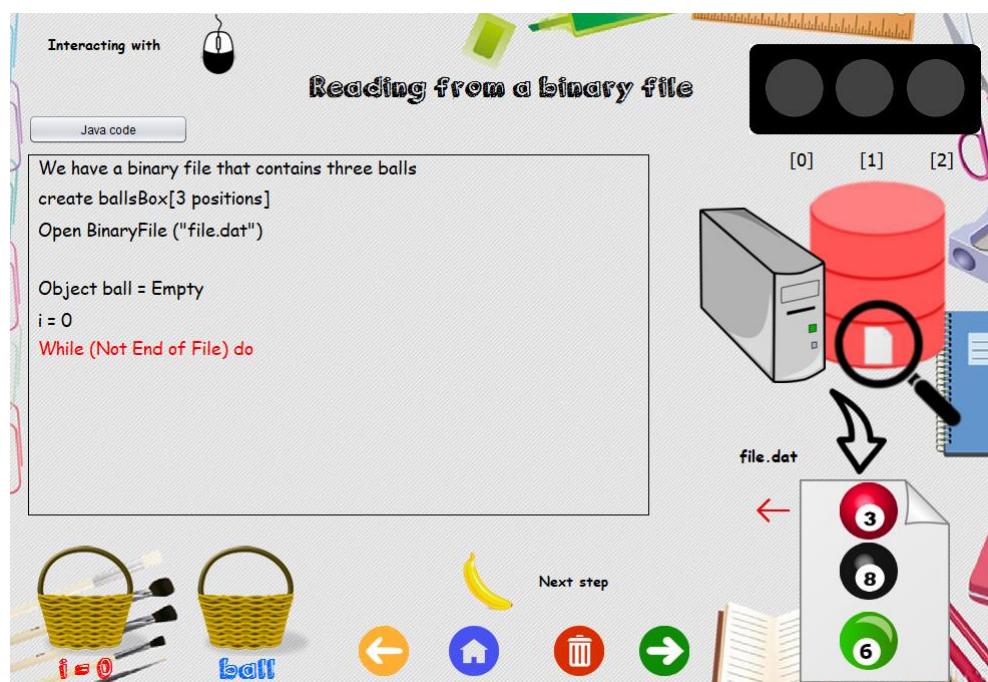


Figura 19. Leer de un fichero binario. Estado inicial.

Fuente: Elaboración propia.

En la Figura 20, se puede observar que se ha guardado la primera bola del fichero en la variable “*ball*” en memoria para posteriormente añadirla al array y que se ha incrementado el valor de la variable “*i*”.

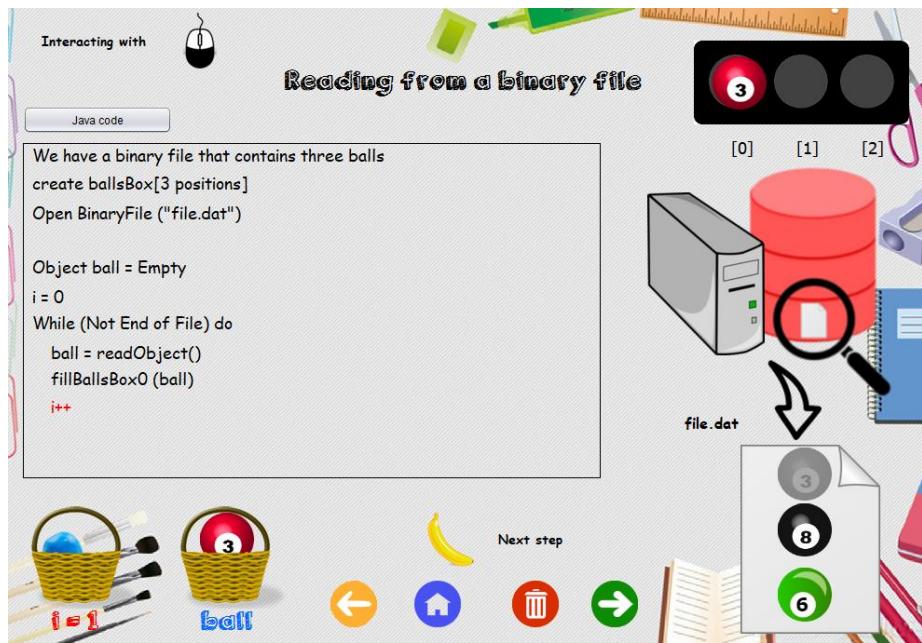


Figura 20. Leer de un fichero binario. Primer valor leído.

Fuente: Elaboración propia.

En la Figura 21, se puede observar que ya se han leido todos los datos del fichero y que se han almacenado todas las bolas en el array. Además se visualiza que la variable iteradora ha aumentado hasta impedir que se produzcan más iteraciones del bucle.

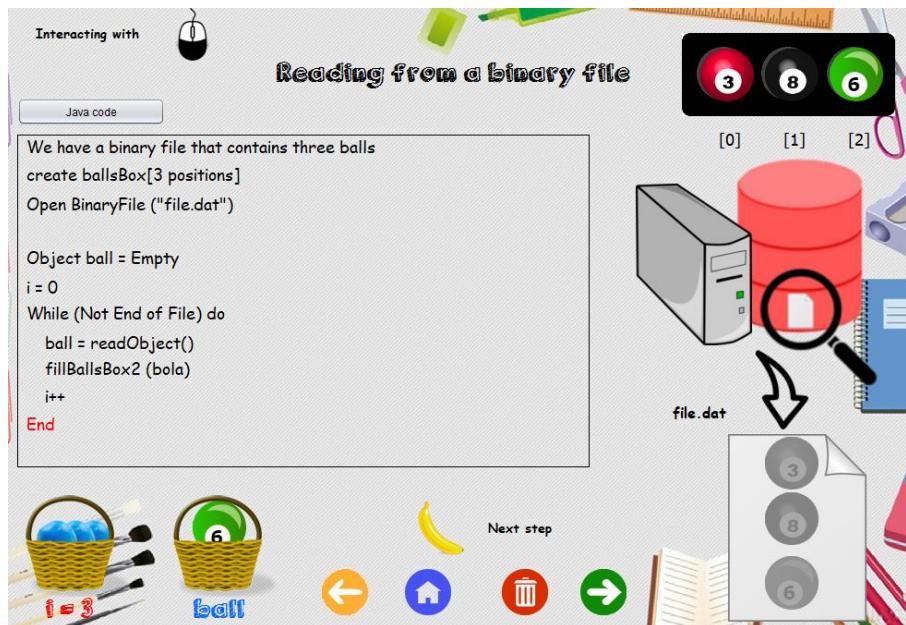


Figura 21. Leer de un fichero binario. Todos los valores leídos.

Fuente: Elaboración propia.

5.10. Leer de un fichero binario 2

En esta pantalla pone un ejemplo de lectura sobre ficheros binarios para imprimir por pantalla cada propiedad del objeto leido de fichero.

Para este caso, los objetos que se encuentran almacenados en el fichero son frutas. Las frutas tienen dos propiedades: el color y el tipo. Para almacenar cada objeto leido del fichero se usa una variable “*fruit*” que se representa mediante una cesta.

En la Figura 22, se puede observar el estado inicial de la pantalla.

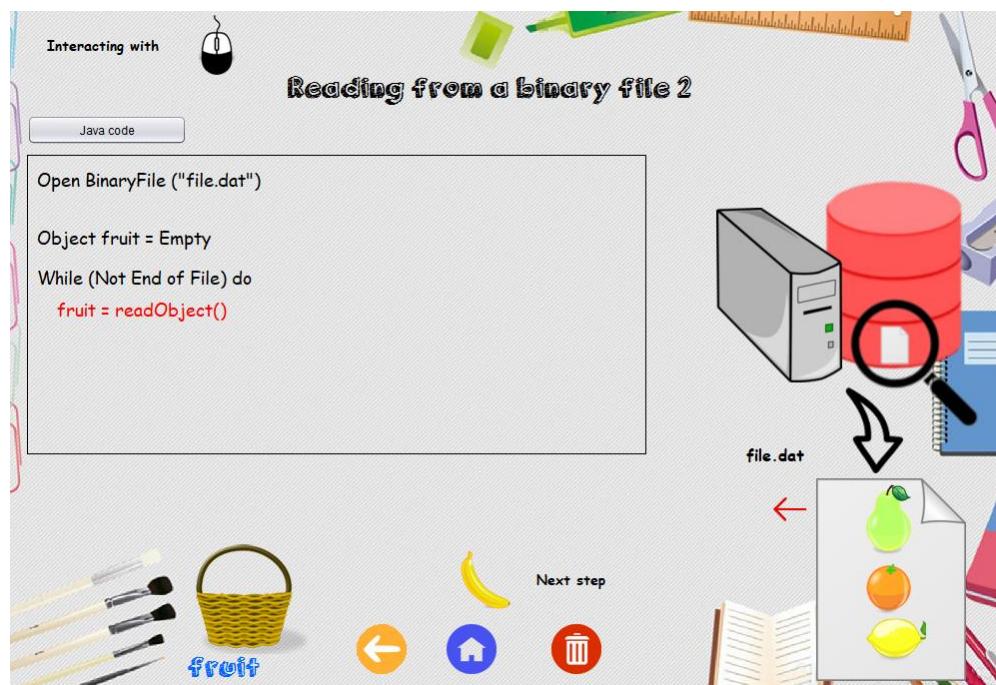


Figura 22. Leer de un fichero binario 2. Estado inicial.

Fuente: Elaboración propia.

En la Figura 23, ya se ha consumido el primer objeto del fichero y se ha guardado su valor en la variable “fruit”. Además se ha impreso por pantalla el tipo de esta fruta y el color.

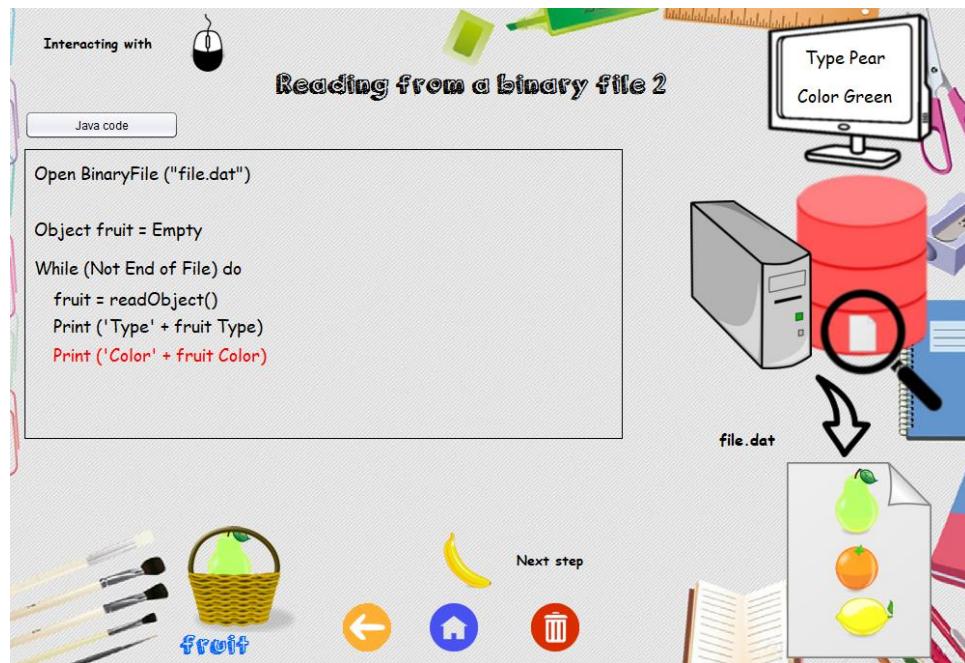


Figura 23. Leer de un fichero binario. Primer valor leído.

Fuente: Elaboración propia.

El mismo proceso se hace para cada uno de los objetos incluidos en el fichero binario.

6. Nueva funcionalidad: Funciones

Para esta nueva funcionalidad los requisitos iniciales han sido los siguientes:

- Explicar qué es una función.
- Mostrar un ejercicio simple en el que se pasan parámetros a la función y devuelve un resultado.
- Mostrar un ejercicio simple en el que se pasan parámetros a la función y no devuelve un resultado.
- Mostrar un ejercicio simple en el que no se pasan parámetros a la función y devuelve un resultado.
- Mostrar un ejercicio simple en el que no se pasan parámetros a la función y no se devuelve un resultado.
- Mostrar un ejercicio de una función más compleja.

En cuanto al diseño y a los casos de uso, nos encontramos con la misma situación que se explica en la sección 5.

6.1. Funciones: ¿qué es una función?

Tras añadir la nueva categoría de funciones era necesario crear una pantalla en la que se explicara de forma breve y sencilla qué es una función. Por ello, en esta pantalla se explica que una función es una pieza de código que se usa para resolver operaciones. Las funciones pueden recibir o no parámetros y además pueden devolver o no un resultado.

En esta pantalla se ponen dos ejemplos de funciones. La función sumar que recibe dos numeros, los suma y devuelve el resultado, y la función juntar que recibe dos letras, las junta e imprime el resultado por pantalla.

Como se puede observar en la Figura 24, se resalta cada parte de la función. El color azul indica el tipo del valor que se devuelve (resultado), el color rojo

indica el nombre de la función, el color verde los parámetros de la función y el color morado el valor que se devuelve.

The slide has a decorative background featuring school-related icons like a pencil, ruler, and calculator. At the top left, there's a small icon of a computer mouse with the text 'Interacting with'. The main title 'Functions: What is a function?' is centered above a text block. Below the title, there are three bullet points: '- ADD function receive two numbers, add them and return the solution.', '- JOIN function receive two letters, join them and print them together.', and 'In addition, not all functions return a solution. Function examples:' followed by the code example. The code example is enclosed in a box:

```
Number Add( number1, number2 ){
    Solution = number1 + number2
    Return Solution
}
```

To the right of the code, there's a vertical legend:

- Type of value to return
- Function name
- Function parameters
- Return value

At the bottom of the slide, there are several navigation icons: a left arrow, a home icon, a trash can icon, a right arrow, and an open book icon. The word 'Next step' is positioned above the right arrow icon.

Figura 24. ¿Qué es una función? Ejemplo de función suma.

Fuente: Elaboración propia.

6.2. Funciones: parámetros y valor devuelto

En esta pantalla se proponen dos funciones que siguen los ejemplos de la pantalla anterior. La función sumarD (addR) recibe dos números, los suma y devuelve el resultado. Por otro lado, la función juntarD (joinR) recibe dos letras, las junta y devuelve la palabra que resulta de juntarlas.

En esta pantalla el usuario se encarga de introducir los parámetros de la función haciendo click en las frutas (que tienen un numero asociado). Además, como se puede observar en la Figura 25, se resalta en los colores explicados anteriormente las partes de la función. Por último, también tenemos dos cestas que representan la variable resultado (valor que se devuelve) y la variable solución (variable que se obtiene del valor devuelto por la función) en memoria.

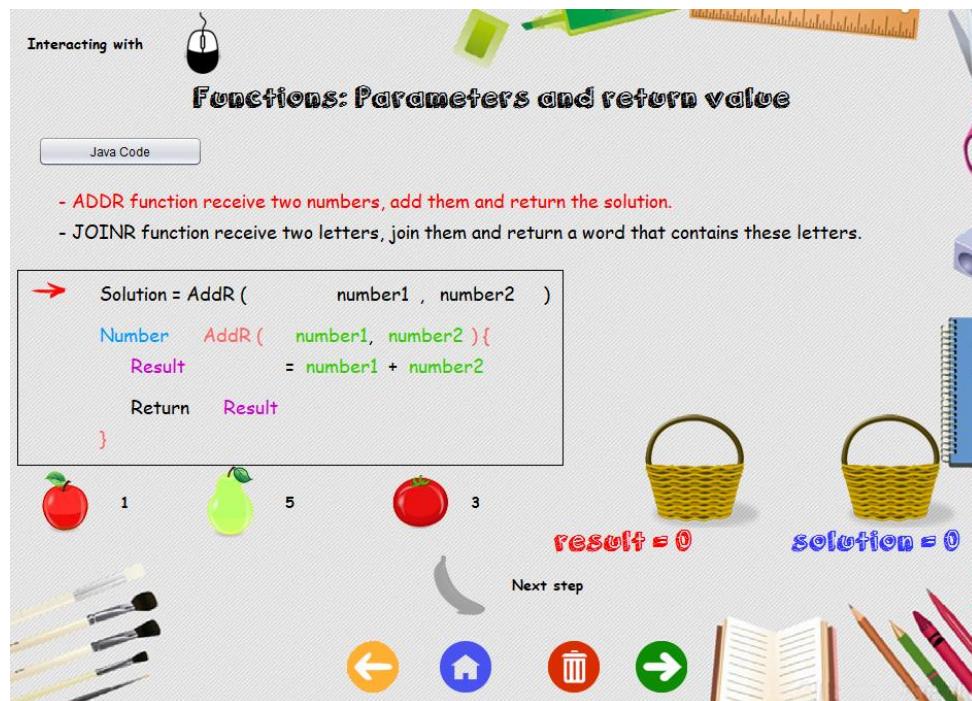


Figura 25. Funciones: Parámetros y valor devuelto. Estado inicial.

Fuente: Elaboración propia.

En la Figura 26, se han introducido los parámetros y se ha devuelto el resultado. Se puede observar también que las cestas ahora almacenan los valores correspondientes.

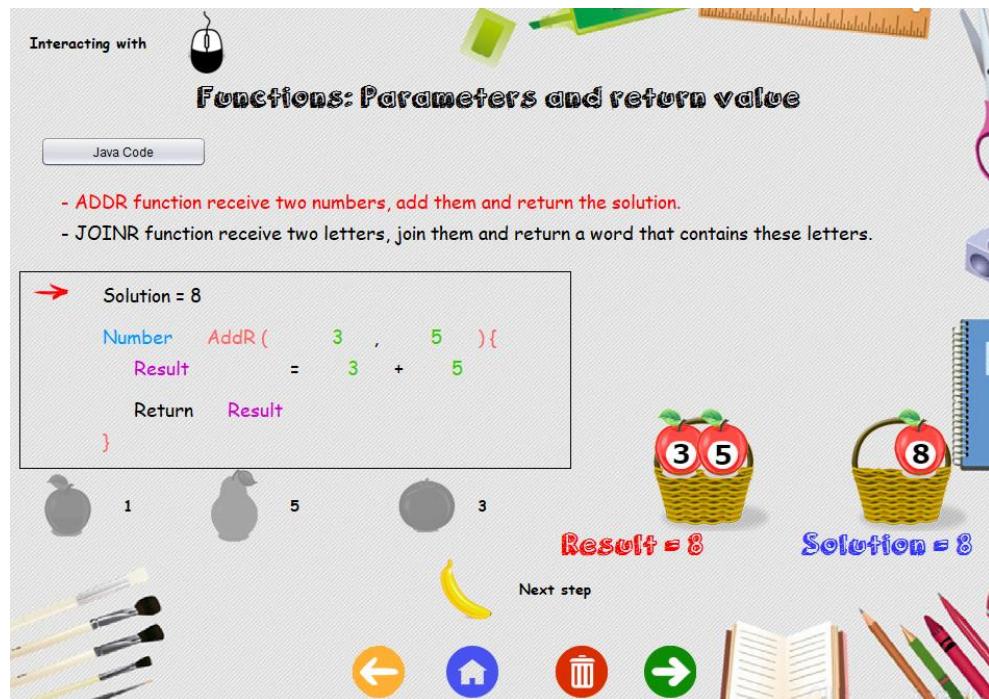


Figura 26. Funciones: Parámetros y valor devuelto. Valor devuelto y valores en las cestas (sumarD).

Fuente: Elaboración propia.

En la Figura 27, se han introducido los parámetros y se ha devuelto el resultado para la función juntarR (joinR) y las cestas almacenan los valores correspondientes.

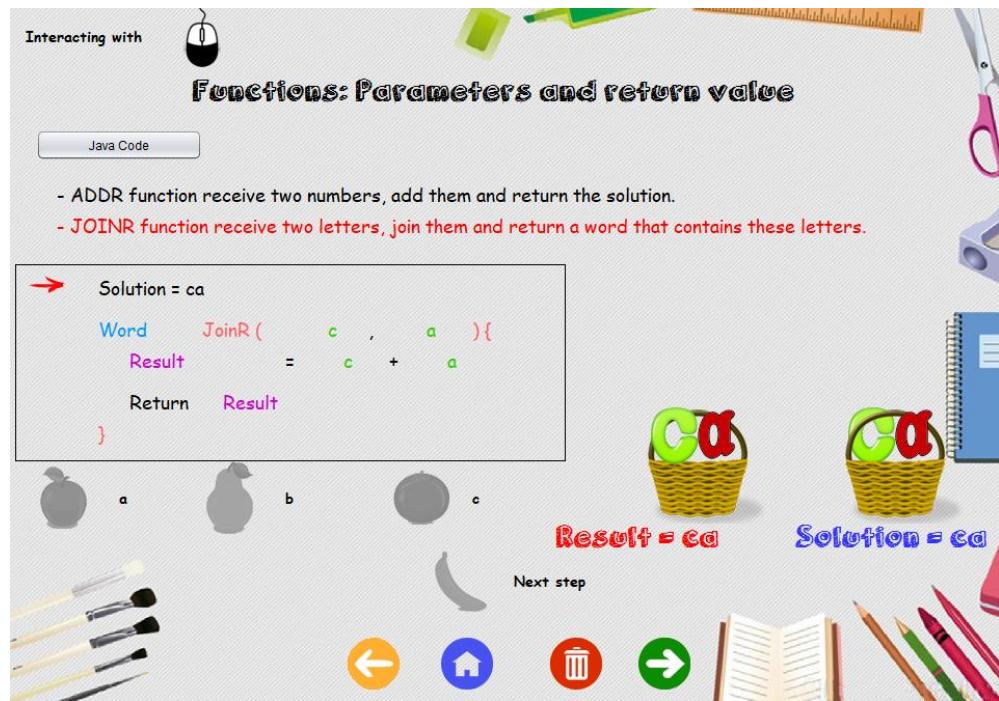


Figura 27. Funciones: Parámetros y valor devuelto. Valor devuelto y valores en las cestas (juntarD).

Fuente: Elaboración propia.

6.3. Funciones: parámetros sin valor devuelto

En esta pantalla se proponen las dos funciones anteriores, pero con un comportamiento distinto. En este caso, la función sumarP (addP) recibe dos números, los suma e imprime el resultado por pantalla, y la función juntarP (joinP) recibe dos letras, las junta e imprime la palabra resultado de juntarlas.

En esta pantalla los parámetros son introducidos por el usuario al igual que en la pantalla anterior.

Como se puede observar en la Figura 28, solo hay una cesta la cual representa el almacenamiento en memoria de la variable resultado.

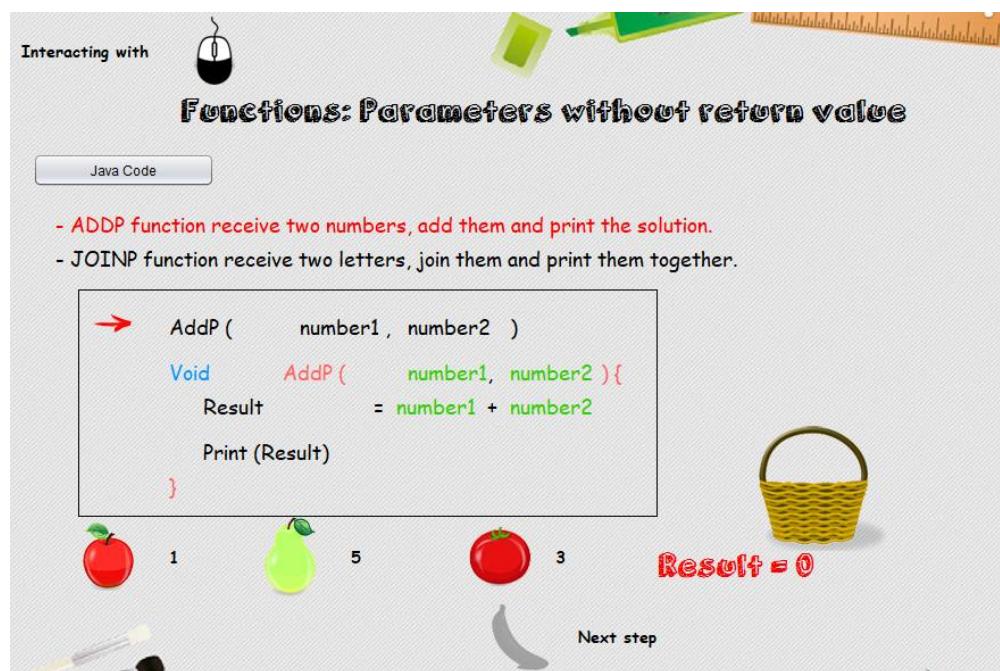


Figura 28. Funciones: Parámetros sin valor devuelto. Estado inicial-

Fuente: Elaboración propia.

En la Figura 29, se puede observar que la cesta almacena el valor del resultado de aplicar la suma a los dos números obtenidos de los parámetros y en la pantalla se ha impreso dicho resultado.

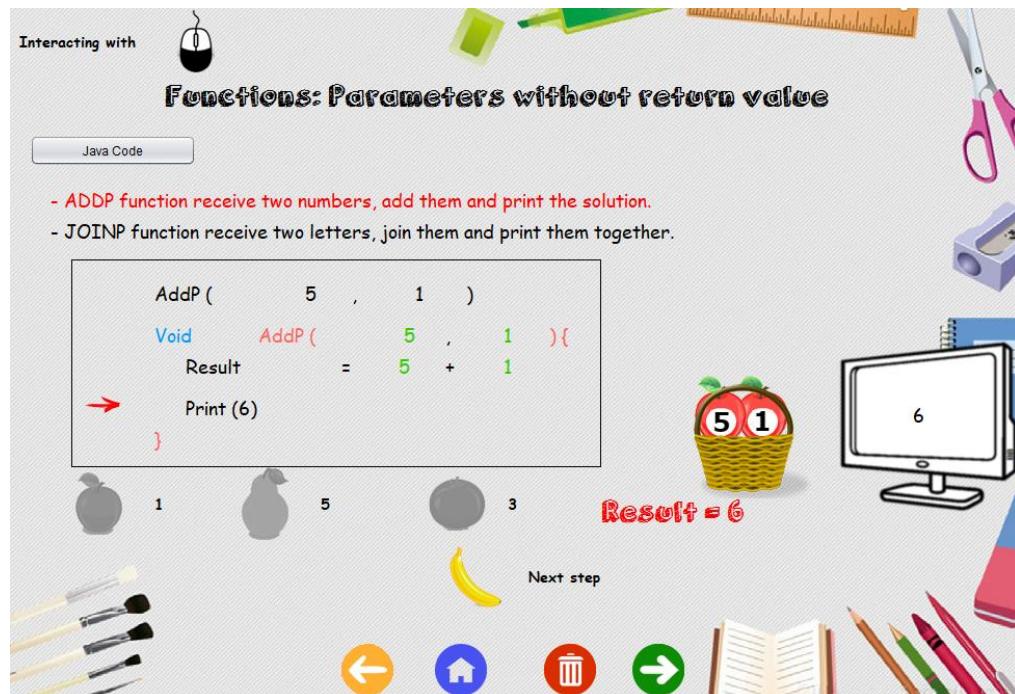


Figura 29. Funciones: Parámetros sin valor devuelto. Resultado en pantalla y valor en la cesta (sumarP).

Fuente: Elaboración propia.

En la Figura 30, se representa el estado final de la pantalla en la cual se almacena en la cesta el valor obtenido al juntar las dos letras introducidas como parámetros y se imprime por pantalla dicho valor.

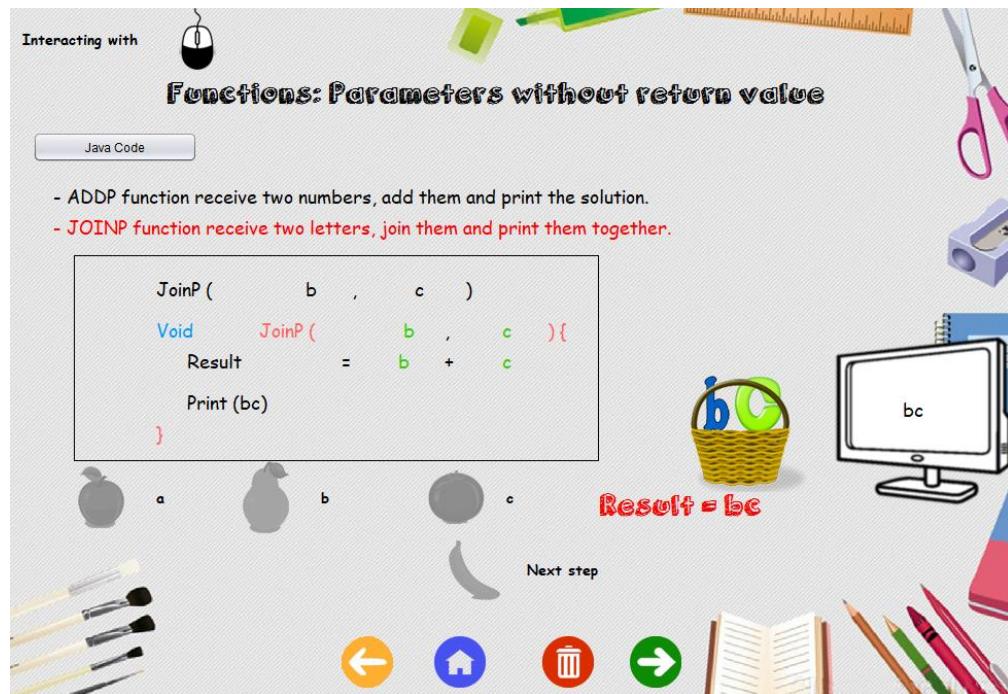


Figura 30. Funciones: parámetros sin valor devuelto. Resultado en la pantalla y valor en la cesta (juntarP).

Fuente: Elaboración propia.

6.4. Funciones: valor devuelto sin parámetros

En esta pantalla se proponen las funciones vistas previamente, pero en este caso, cada función tiene un comportamiento distinto. La función sumarDosNumerosD (addTwoNumbersR) se encarga de sumar los números 2 y 6 y devolver el resultado, mientras que la función juntarDosLetrasD (joinTwoLettersR) se encarga de juntar las letras a y b y devolver el resultado.

Como en esta pantalla se están poniendo ejemplos de funciones sin parámetros de entrada, los valores usados en las operaciones están por defecto. Además, nos encontramos con dos cestas, la primera representa el almacenamiento en memoria del resultado calculado en la función, mientras que la segunda representa el almacenamiento en memoria del valor que se devuelve en la función.

Como se puede observar en la Figura 31, se ha realizado la suma y las cestas ahora almacenan el valor calculado y el valor devuelto respectivamente.

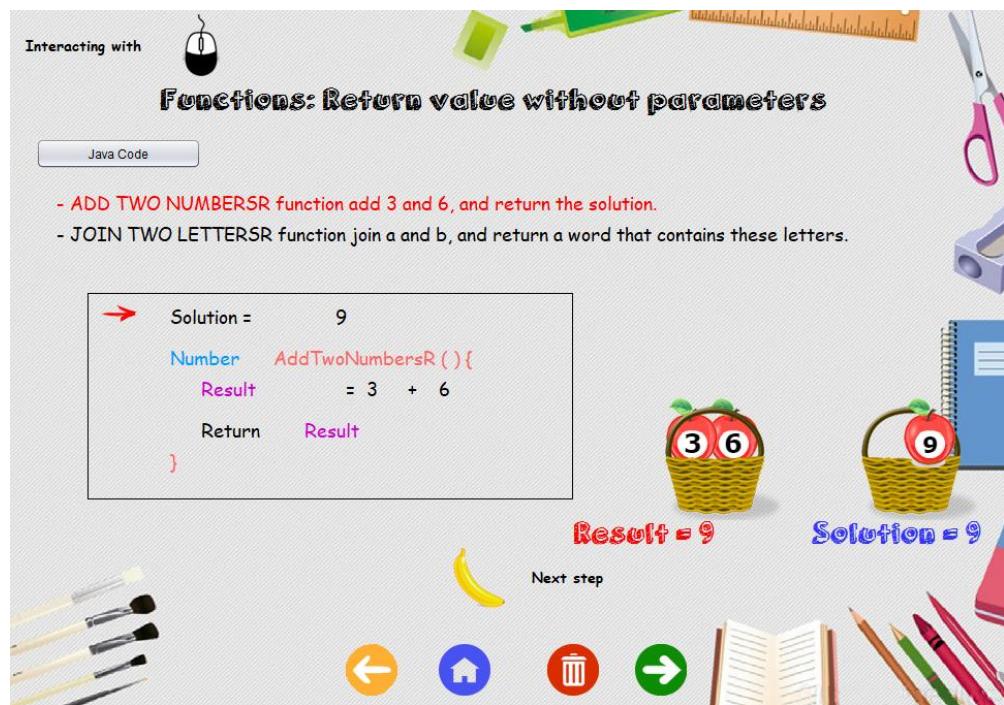


Figura 31. Funciones: valor devuelto sin parámetros. Valor devuelto y valores en las cestas (sumarDosNumerosR).

Fuente: Elaboración propia.

Como se puede observar en Figura 32, se ha realizado la operación que junta las letras (a y b) y las cestas ahora almacenan el valor calculado y el valor devuelto respectivamente.

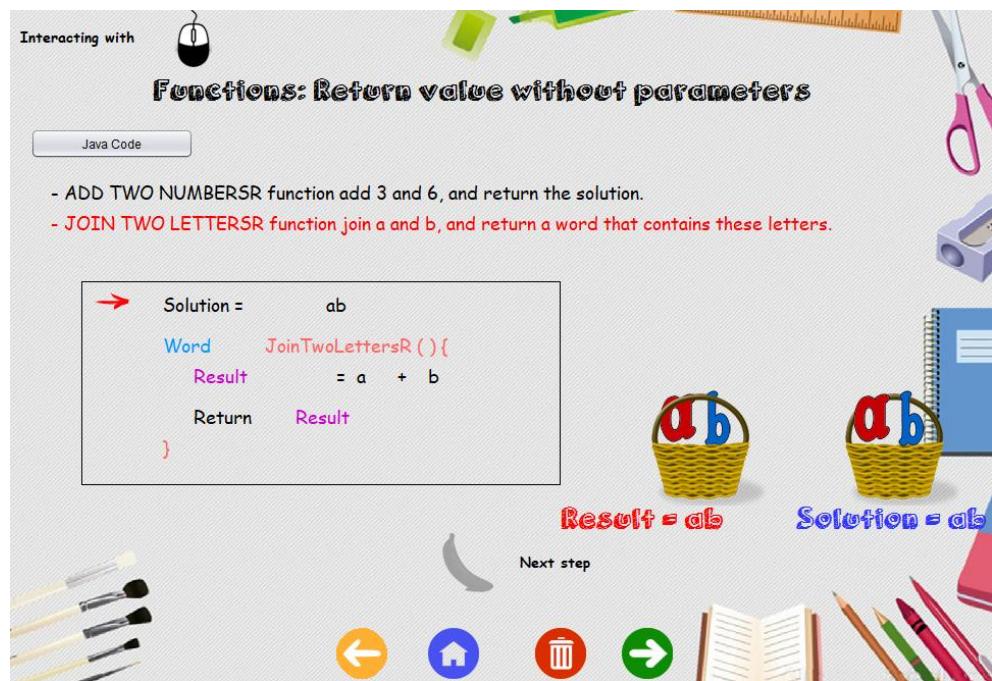


Figura 32. Funciones: valor devuelto sin parámetros. Valor devuelto y valores en las cestas (juntarDosLetrasR).

Fuente: Elaboración propia.

6.5. Funciones: sin parámetros ni valor devuelto

En esta pantalla se proponen las funciones anteriores pero en este caso cada función tiene un comportamiento distinto a los casos anteriores. La función sumarDosNumerosP (addTwoNumbersP) suma los numeros 3 y 6 e imprime el resultado por pantalla, y la función juntarDosLetrasP (joinTwoLettersP) junta las letras a y b e imprime la palabra resultado por pantalla.

Como se están poniendo ejemplos de funciones que no reciben parámetros, los valores usados en las operaciones están puestos por defecto. Además, como no se devuelve ningún valor, solamente hay una cesta que representa el almacenamiento en memoria del valor calculado en la función.

Como se puede observar en la Figura 33, se ha realizado la suma de los números (3 y 6), la cesta almacena el valor obtenido de dicha suma y se ha impreso el resultado por pantalla.

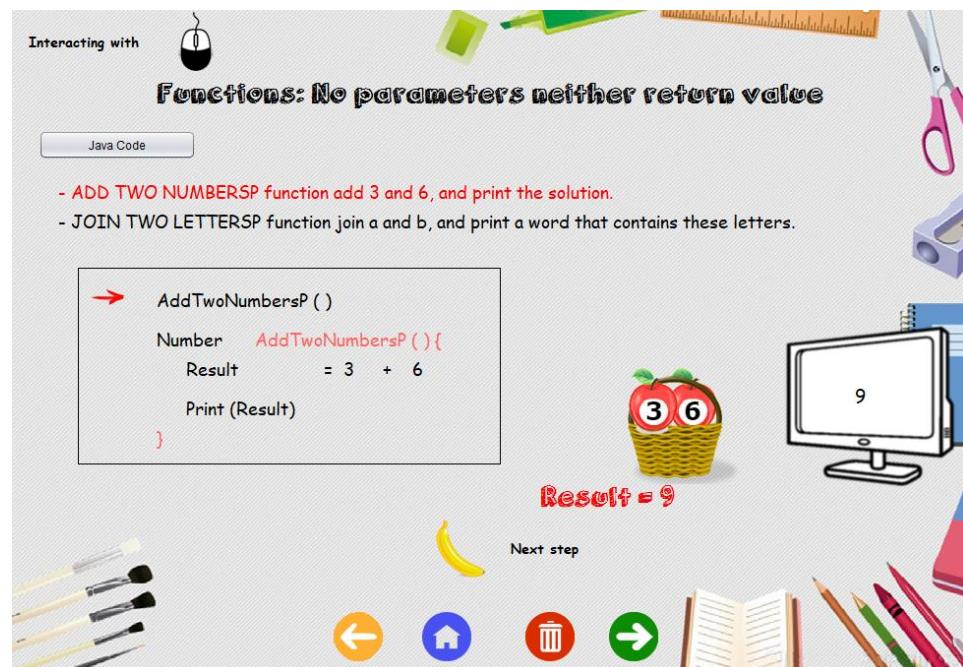


Figura 33. Funciones: sin parámetros ni valor devuelto. Resultado impreso en pantalla y valor en la cesta (sumarDosNumerosP).

Fuente: Elaboración propia.

Como se puede observar en la Figura 34, se ha realizado la operación que junta las letras (a y b), la cesta almacena el valor obtenido de dicha operación y se ha impreso el resultado por pantalla.

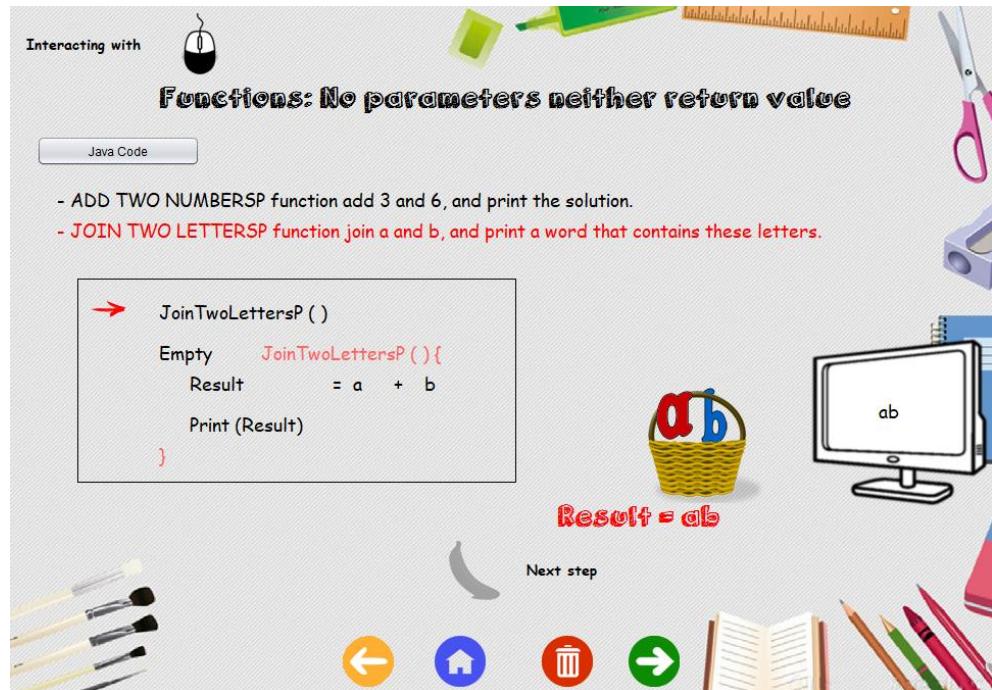


Figura 34. Funciones: sin parámetros ni valor devuelto. Resultado impreso en pantalla y valor en la cesta (juntarDosLetrasP).

Fuente: Elaboración propia.

6.6. Funciones: una función más completa

En esta pantalla se propone una función más completa y compleja para aplicar conocimientos obtenidos en otras categorías de la aplicación. La función propuesta sumarNumerosBolas recibe un array de bolas de billar, suma los numeros de las bolas y devuelve el resultado de la suma.

Como se puede observar en la Figura 35, hay cuatro cestas. Una cesta representa el almacenamiento en memoria del resultado obtenido en la función, otra el almacenamiento en memoria del valor devuelto en la función, otra el almacenamiento en memoria de la variable iteradora y la última el almacenamiento en memoria de la longitud del array de bolas.

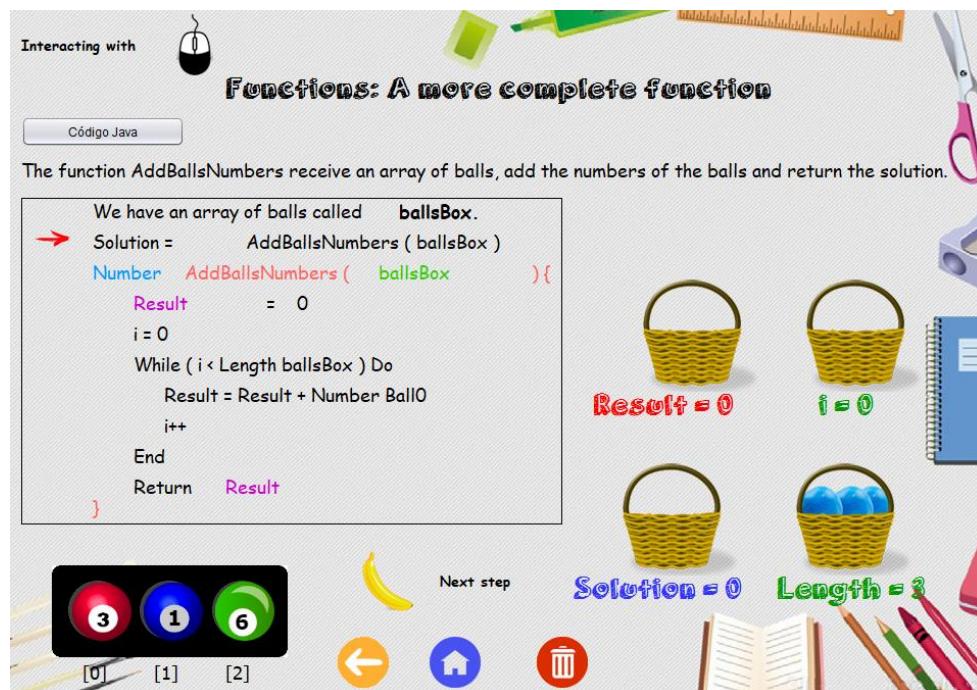


Figura 35. Funciones: una función más completa. Estado inicial.

Fuente: Elaboración propia.

En la Figura 36, se puede observar que se ha obtenido el valor de la primera bola del array y se le ha sumado al valor del resultado (primera iteración completada). Las cestas se han actualizado además con sus valores correspondientes, exceptuando la cesta de la solución puesto que aún no se ha devuelto el resultado.

```

Interacting with
Código Java
Functions: A more complete function

The function AddBallsNumbers receive an array of balls, add the numbers of the balls and return the solution.

We have an array of balls called ballsBox.
Solution = AddBallsNumbers ( ballsBox )
Number AddBallsNumbers ( ballsBox ) {
    Result = 0
    i = 0
    → While ( i < Length ballsBox ) Do
        Result = Result + Number Ball(i)
        i++
    End
    Return Result
}

```

Figura 36. Funciones: una función más completa. Primera iteración.

Fuente: Elaboración propia.

En la Figura 37, se ha obtenido el valor de la segunda bola del array y se le ha sumado al valor de resultado (segunda iteración completada). Las cestas se han actualizado con sus valores correspondientes, exceptuando la cesta de la solución puesto que aún no se ha devuelto el resultado.

Interacting with

Functions: A more complete function

Código Java

The function AddBallsNumbers receive an array of balls, add the numbers of the balls and return the solution.

```
We have an array of balls called    ballsBox.  
Solution =      AddBallsNumbers ( ballsBox )  
Number  AddBallsNumbers (   ballsBox    ) {  
    Result      =  0  
    i = 0  
    → While ( i < Length ballsBox ) Do  
        Result = Result + Number Ball1  
        i++  
    End  
    Return   Result  
}
```

Figura 37. Funciones: una función más completa. Segunda iteración.

Fuente: Elaboración propia.

Por último, en la Figura 38 se ha obtenido el valor de la tercera bola del array y se le ha sumado al valor del resultado (tercera iteración completada). También se ha salido del bucle y se ha retornado el valor calculado. Por tanto, ahora todas las cestas tienen sus valores correspondientes.

Interacting with

Functions: A more complete function

Código Java

The function AddBallsNumbers receive an array of balls, add the numbers of the balls and return the solution.

We have an array of balls called `ballsBox`.

```

→ Solution = 10
Number AddBallsNumbers ( ballsBox ) {
    Result = 0
    i = 0
    While ( i < Length ballsBox ) Do
        Result = Result + Number Ball2
        i++
    End
    Return Result
}

```

Result = 10 i = 3

Next step

← Home Delete

Solution = 10 Length = 3

Figura 38. Funciones: una función más completa. Estado final.

Fuente: Elaboración propia.

7. Nueva funcionalidad: Recursividad

Para esta nueva funcionalidad los requisitos iniciales han sido los siguientes:

- Explicar qué son las funciones recursivas. Recursividad lineal y recursividad por la cola.
- Mostrar un ejercicio de función recursiva lineal para una operación de suma.
- Mostrar un ejercicio de función recursiva lineal para calcular el factorial de un número.
- Mostrar un ejercicio de función recursiva por la cola para una operación de suma.
- Mostrar un ejercicio de función recursiva por la cola para calcular el factorial de un número.

En cuanto al diseño y a los casos de uso, nos encontramos con la misma situación que se explica en la sección 5.

7.1. Funciones recursivas

En esta pantalla se explica de una forma sencilla qué es una función recursiva. Una función recursiva es una función que se llama a sí misma, directamente o a través de otra función.

La recursividad tiene dos tipos de casos, el caso recursivo y el caso base. El caso recursivo es el caso más complejo y se divide en casos más pequeños y simples. El caso base es el caso más sencillo y es la última llamada que se realiza a la función que devuelve una solución directamente.

Por último, se explican los dos tipos de recursividad existentes. La recursividad lineal, en la que cada llamada recursiva genera una llamada recursiva como mucho y el resultado lo devuelve la primera llamada recursiva. Y la

recursividad por la cola, en la que el resultado se devuelve en la última llamada y no se realizan operaciones.

También se pone un ejemplo simple de una función recursiva, en este caso, la función sumar que recibe un número y calcula la suma de ese número más todos sus antecesores. Por ejemplo, si el número recibido como parámetro es 6, el resultado sería $6 + 5 + 4 + 3 + 2 + 1 = 21$.

Si el número es igual a uno estamos en el caso base y por tanto se devuelve una solución directa, en este caso un uno. En cualquier otro caso, se hace una llamada recursiva a la función sumar pero restándole uno al número recibido como parámetro. El valor devuelto en este caso recursivo es el valor devuelto al hacer la llamada recursiva explicada anteriormente y sumarle el valor del número recibido como parámetro. En la Figura 39 se puede observar cómo queda esta pantalla.

The slide has a decorative top border featuring school-related icons like a pencil, ruler, and eraser. The title 'Recursive functions' is centered at the top. On the left, there's a small icon of a computer mouse with the text 'Interacting with'. The main content area contains the following text and code:

A function is recursive if it calls itself, directly or through another function.

There are two cases of recursion:

- Recursive case: the most difficult case and it is divided into smaller cases.
- Base case: the last call to the function that returns directly a solution.

Recursion types:

- Lineal recursion: each recursive call generates one recursive call as much.
- Tail recursion: the solution is returned in the last call. No operations are performed.

```
Number    Add(  number1  ){
    If (number1 == 1)
        Return 1
    End-If
    Return   Add (number1 - 1) + number1
}
```

Base case
Recursive case

Figura 39. Explicación de las funciones recursivas.

Fuente: Elaboración propia.

7.2. Recursividad lineal: función sumar

En esta pantalla se pone un ejemplo para explicar cómo funciona la recursividad lineal.

Para ello se utiliza la función sumar que recibe dos números y devuelve el resultado de sumarlos.

Como se puede observar en la Figura 40, tras varias ejecuciones se ha llegado a un caso base, que en este caso devuelve el valor 2. En la parte izquierda nos encontramos con el pseudocódigo de la función, en la parte derecha nos encontramos con varias cajas las cuales representan cada paso recursivo y también hay una cesta la cual sirve para representar el almacenamiento de la solución en memoria.

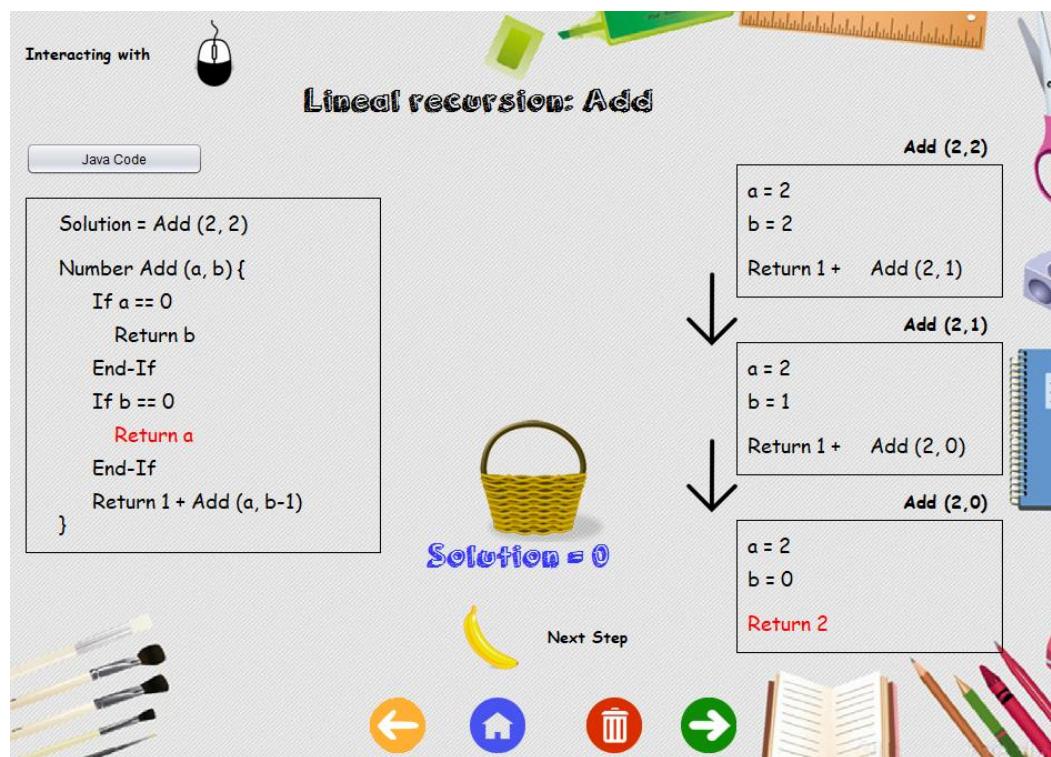


Figura 40. Recursividad lineal: función sumar.

Fuente: Elaboración propia.

En la figura 41 se puede observar que se ha devuelto el valor del caso base, y por tanto para la llamada recursiva en la que nos encontramos ahora se puede devolver también un valor, en este caso se devuelve el valor 3 ($1 + 2$).

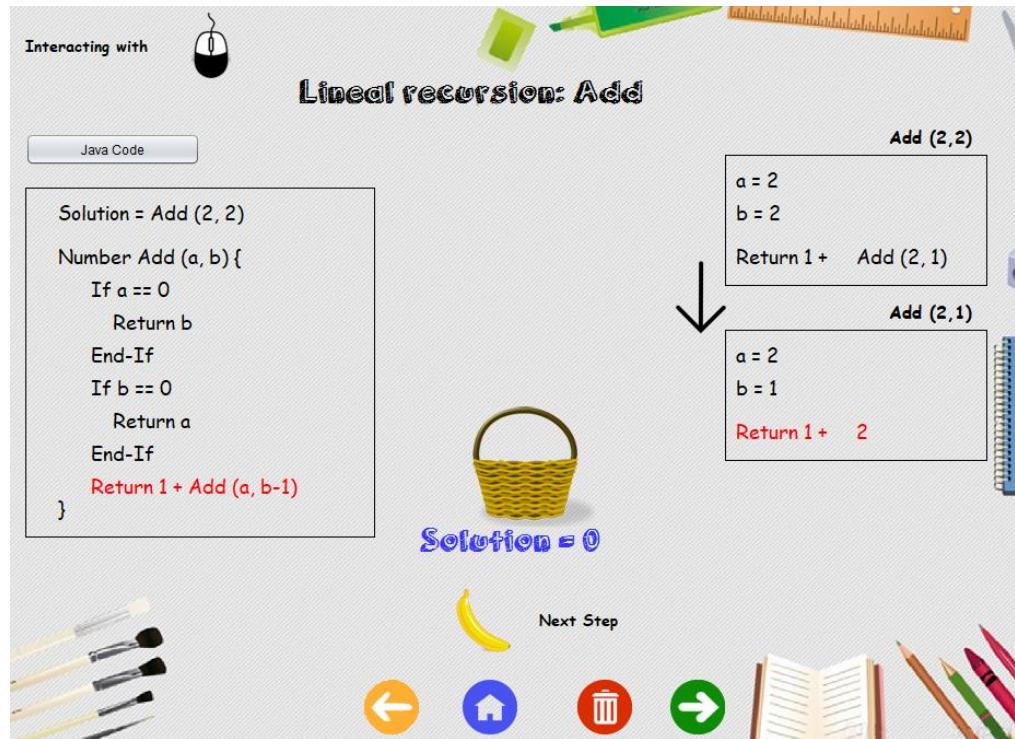


Figura 41. Recursividad lineal: función sumar. Segunda llamada recursiva.

Fuente: Elaboración propia.

En la Figura 42 se puede observar que se ha devuelto el valor de la llamada recursiva y por tanto ahora nos encontramos en la primera llamada, que puede devolver el resultado, en este caso el resultado es 4 ($1 + 3$). Finalmente, este valor devuelto por la primera llamada se almacena en la variable solución como se puede observar a su vez en la Figura 43.

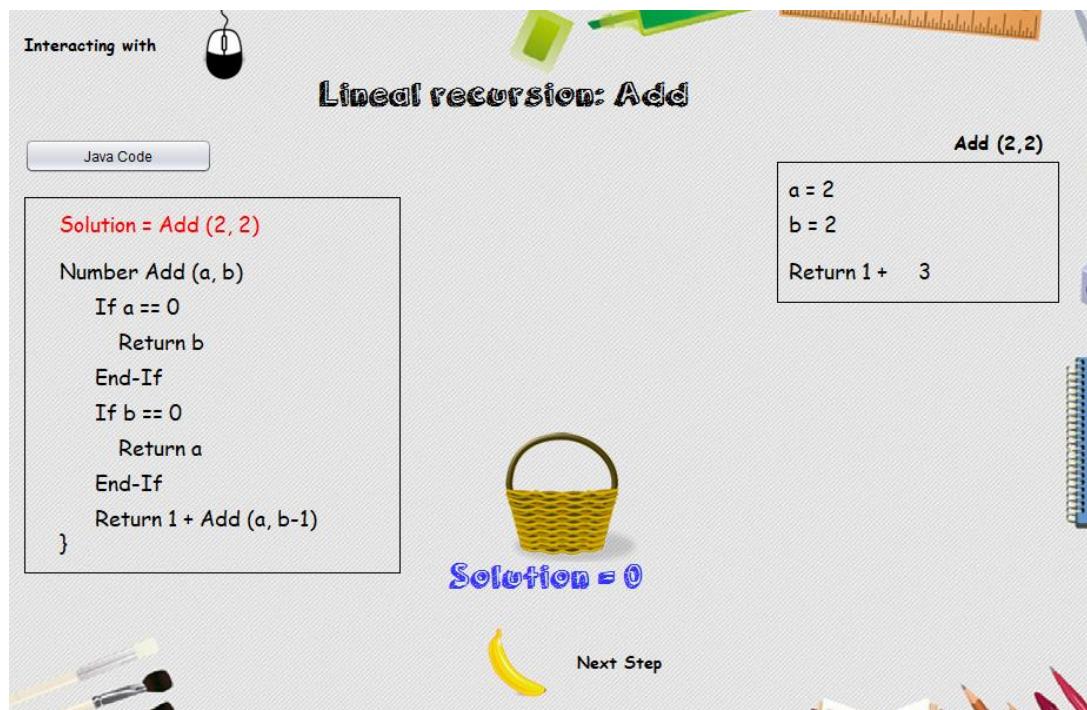


Figura 42. Recursividad lineal: función sumar. Valor devuelto en la primera llamada.

Fuente: Elaboración propia.

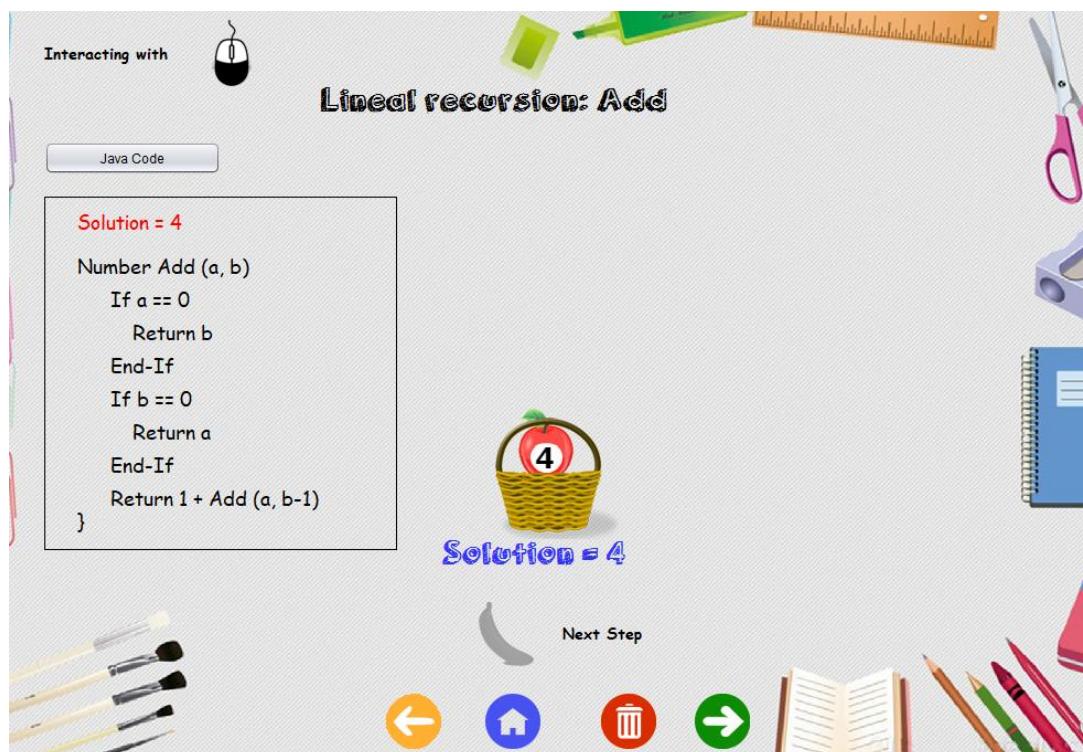


Figura 43. Recursividad lineal: función sumar. Resultado almacenado.

Fuente: Elaboración propia.

7.3. Recursividad lineal: función factorial

En esta pantalla se pone otro ejemplo más avanzado de recursividad lineal como puede ser la función factorial. Esta función recibe un numero y devuelve la multiplicación de este número y sus antecesores. Por ejemplo, si recibimos un 3 como parámetro la solución sería $3 * 2 * 1 = 6$.

Como se puede observar en la Figura 44, en la parte derecha tenemos todas las cajas que se corresponden a cada paso recursivo realizado. Tras todas las llamadas recursivas se ha llegado a un caso base que devuelve el valor 1.

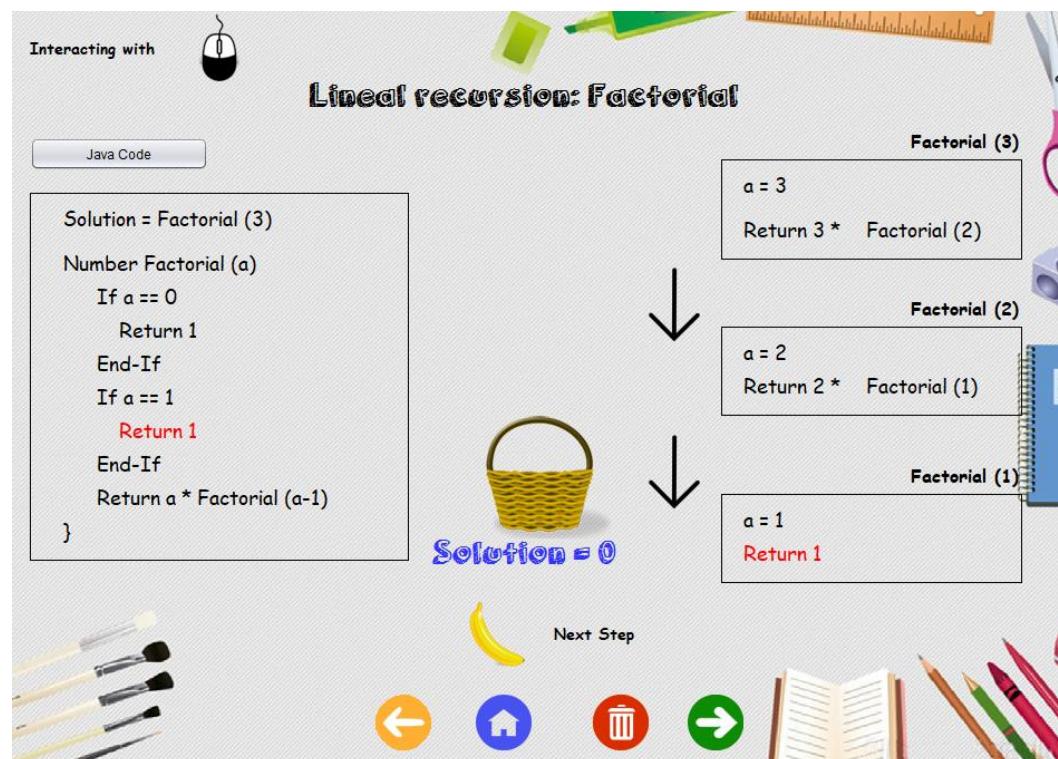


Figura 44. Recursividad lineal: función factorial.

Fuente: Elaboración propia.

Como en el caso de la función sumar, cada llamada recursiva va devolviendo su valor a la llamada recursiva anterior hasta que llegamos a la primera llamada. Como se puede observar en la Figura 45, cada llamada recursiva ha ido devolviendo su valor y ahora nos encontramos en la primera llamada que devolverá un 6 ($3 * 2$).

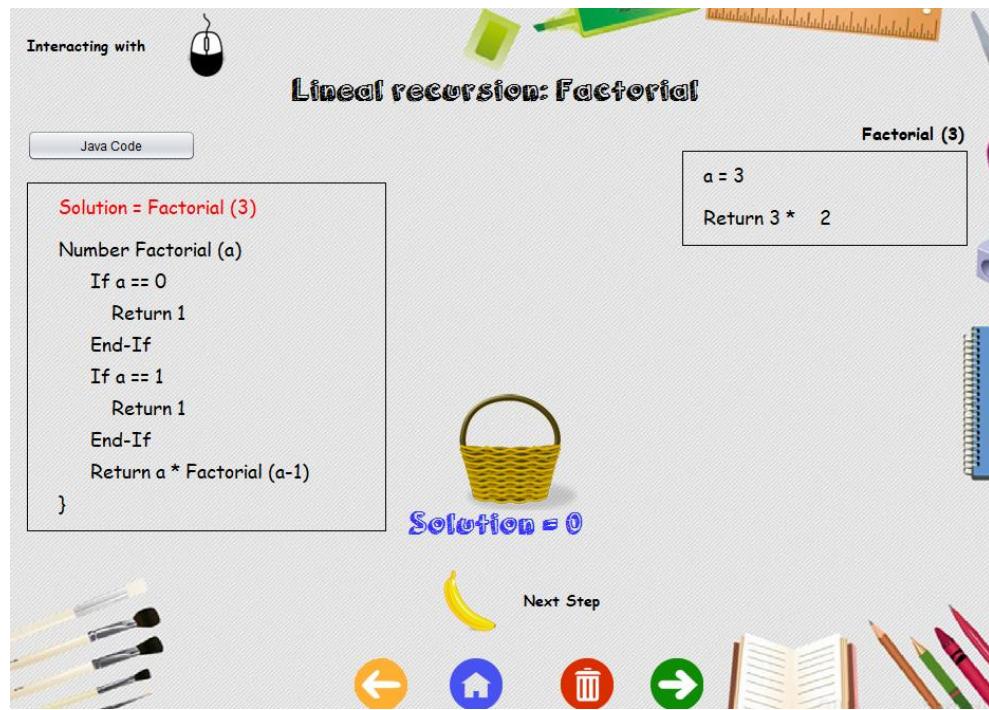


Figura 45. Recursividad lineal: función factorial. Valor devuelto en la primera llamada.

Fuente: Elaboración propia.

Por último el valor devuelto se almacena en la variable solución como se puede comprobar en la Figura 46.

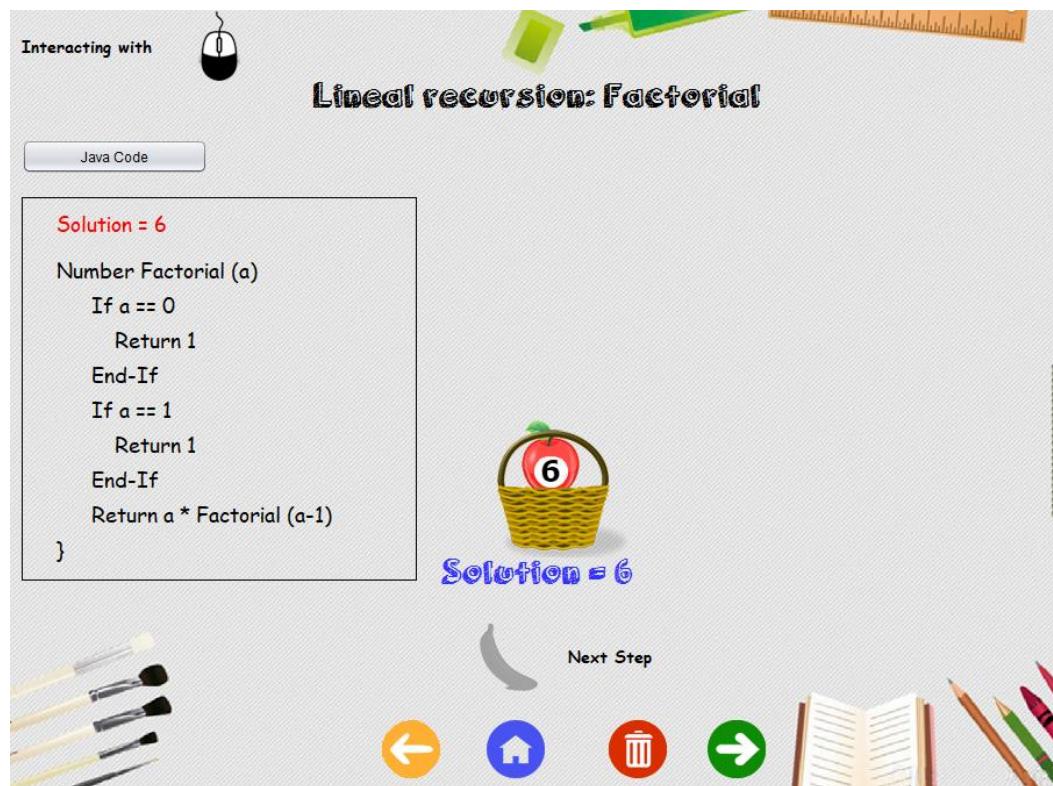


Figura 46. Recursividad lineal: función factorial. Resultado almacenado.

Fuente: Elaboración propia.

7.4. Recursividad por la cola: función sumar

En la recursividad por la cola, no es la primera llamada recursiva la que devuelve el resultado puesto que no se pueden realizar operaciones en los casos recursivos. La última llamada recursiva es la que devuelve el resultado obtenido a partir de un paso recursivo en el cual se devuelve el resultado a partir de un caso base.

Como no se pueden realizar operaciones en los casos recursivos necesitamos una variable que vaya guardando el progreso, por tanto tenemos una primera función **Sumar** (que recibe dos numeros) que llama a la función recursiva **SumarCola** (que recibe esos dos numeros y una variable acumulativa).

En esta pantalla se representa la función sumar utilizando la recursividad por la cola. Como se puede observar en la Figura 47, a la izquierda nos encontramos con el pseudocódigo de la función, a la derecha tenemos una serie de cajas que representan cada paso recursivo y también hay una cesta que representa el valor devuelto por la función almacenado en memoria en la variable solución. En esta figura hemos llegado a un caso base que en este caso devuelve un 3 ($2 + 1$).

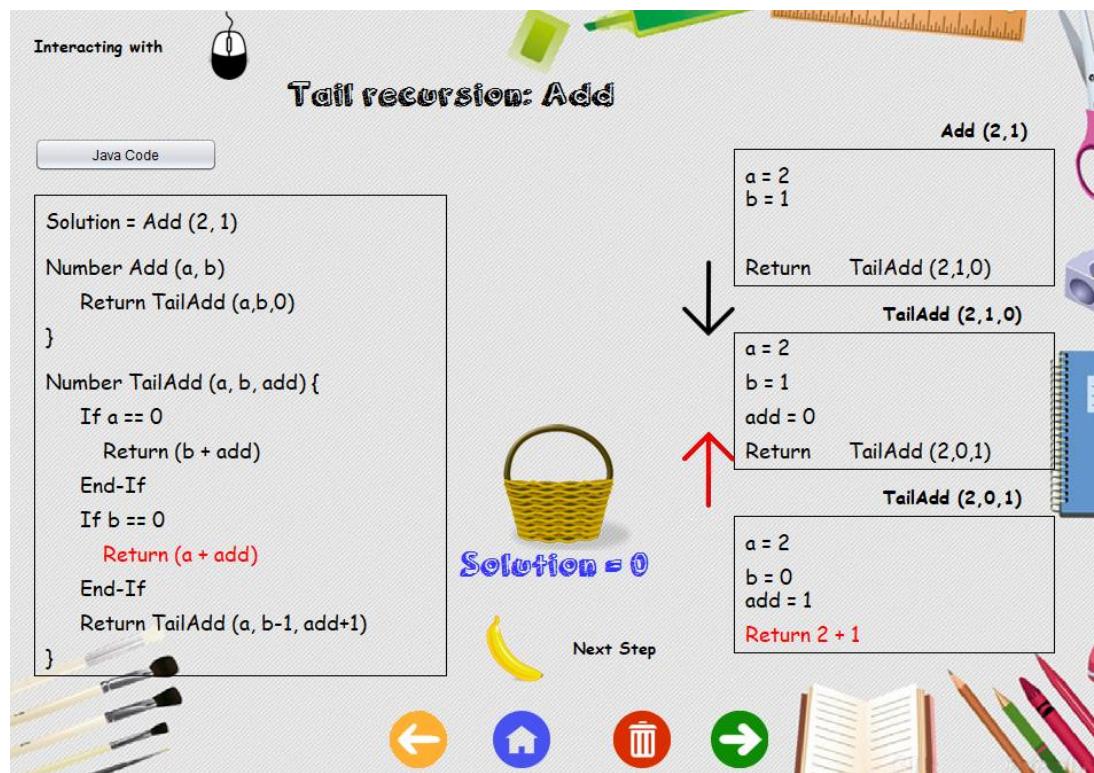


Figura 47. Recursividad por la cola: función sumar.

Fuente: Elaboración propia.

En la Figura 48 se puede observar que la ultima llamada recursiva obtiene el valor devuelto por el caso base, en este caso 3, y devuelve este resultado, eliminando de memoria el resto de llamadas recursivas anteriores.

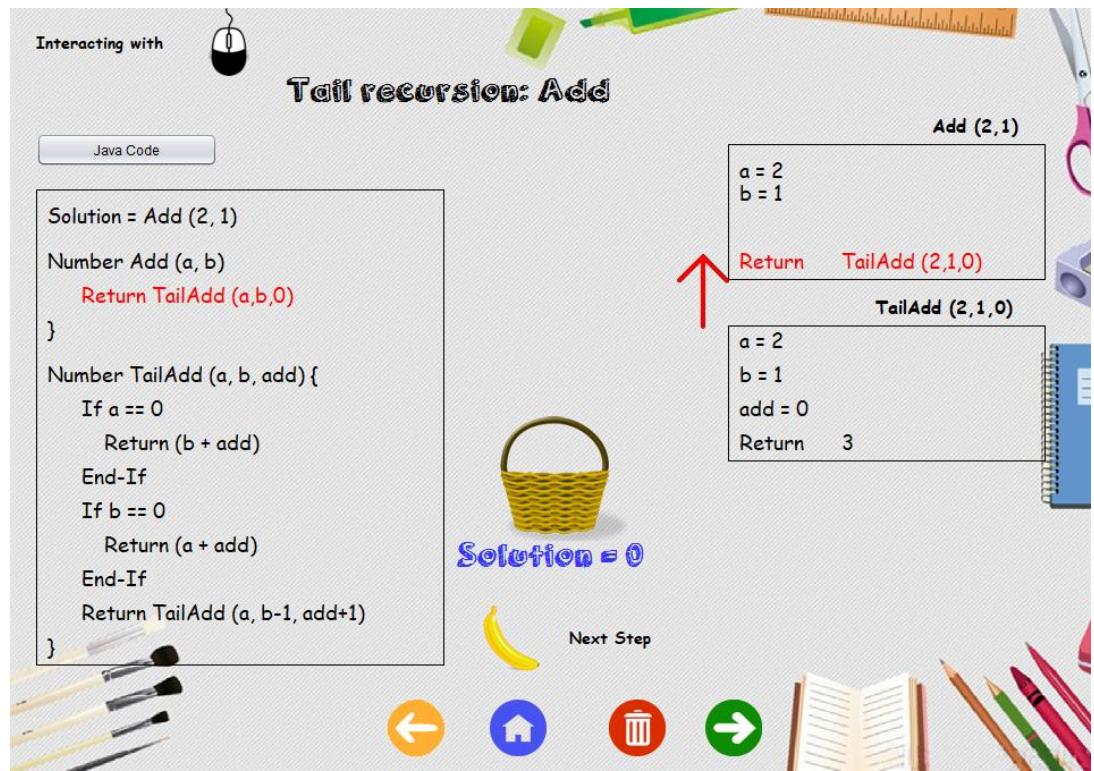


Figura 48. Recursividad por la cola: función sumar. Valor devuelto por la última llamada recursiva.

Fuente: Elaboración propia.

En la Figura 49 la función recursiva ya ha devuelto el resultado (3) y ahora la función que se encarga de llamar a la función recursiva devuelve su valor para almacenarlo en memoria. En la Figura 50 se puede observar el resultado almacenado.

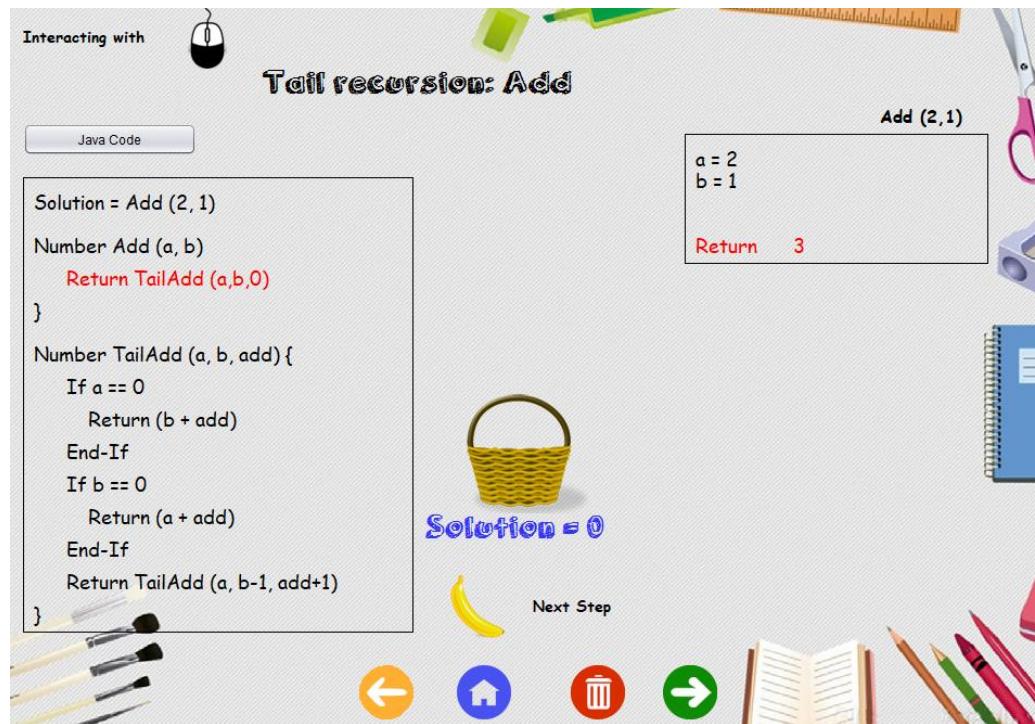


Figura 49. Recursividad por la cola: función sumar. Valor devuelto por la función inicial.

Fuente: Elaboración propia.

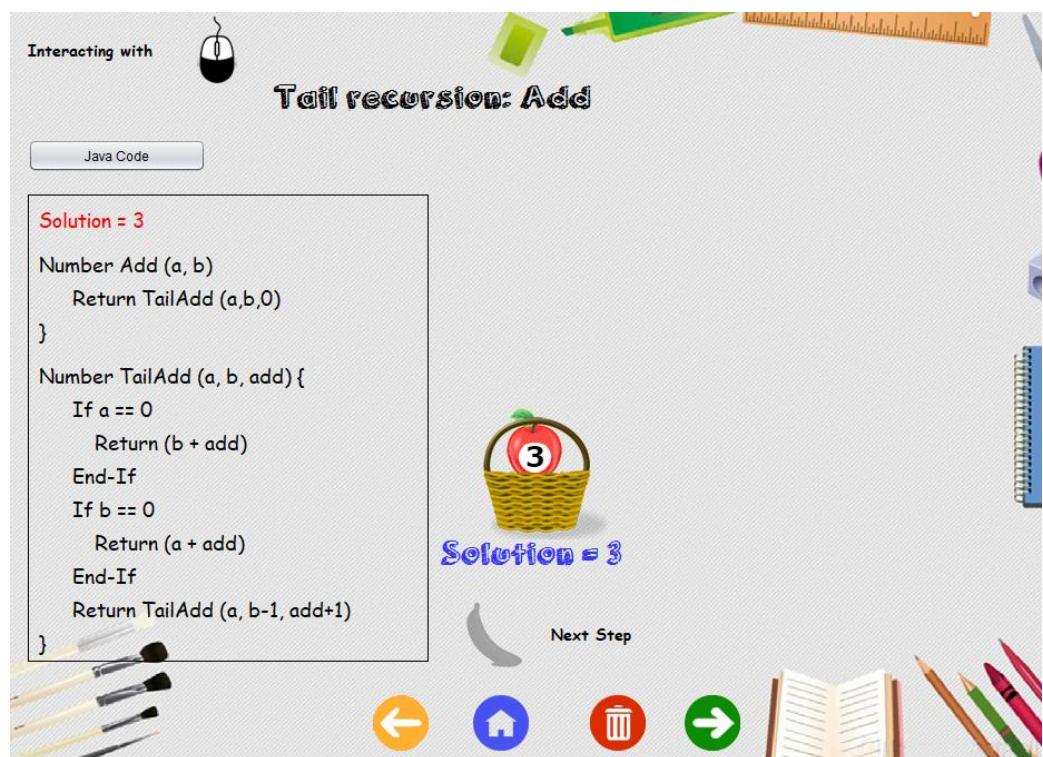


Figura 50. Recursividad por la cola: función sumar. Resultado almacenado.

Fuente: Elaboración propia.

7.5. Recursividad por la cola: función factorial

En esta pantalla se representa otro ejemplo de recursividad por la cola pero en este caso se usa la función factorial. Para ejemplificar esta función, necesitamos una función inicial **Factorial** que recibe como parámetro un número y se encarga de devolver como resultado el factorial de dicho número. Para ello, la función inicial hace una llamada a la función recursiva **FactorialCola** que consta de un caso base y un caso recursivo. A esta función recursiva le llegan como parámetros el número del que obtener el factorial y una variable acumuladora.

Como se puede observar en la Figura 51, a la izquierda tenemos el pseudocódigo de las funciones, a la derecha unas cajas que representan los pasos recursivos terminando en el caso base, y una cesta que representa el almacenamiento en memoria del resultado devuelto. El caso base devuelve el valor 2.

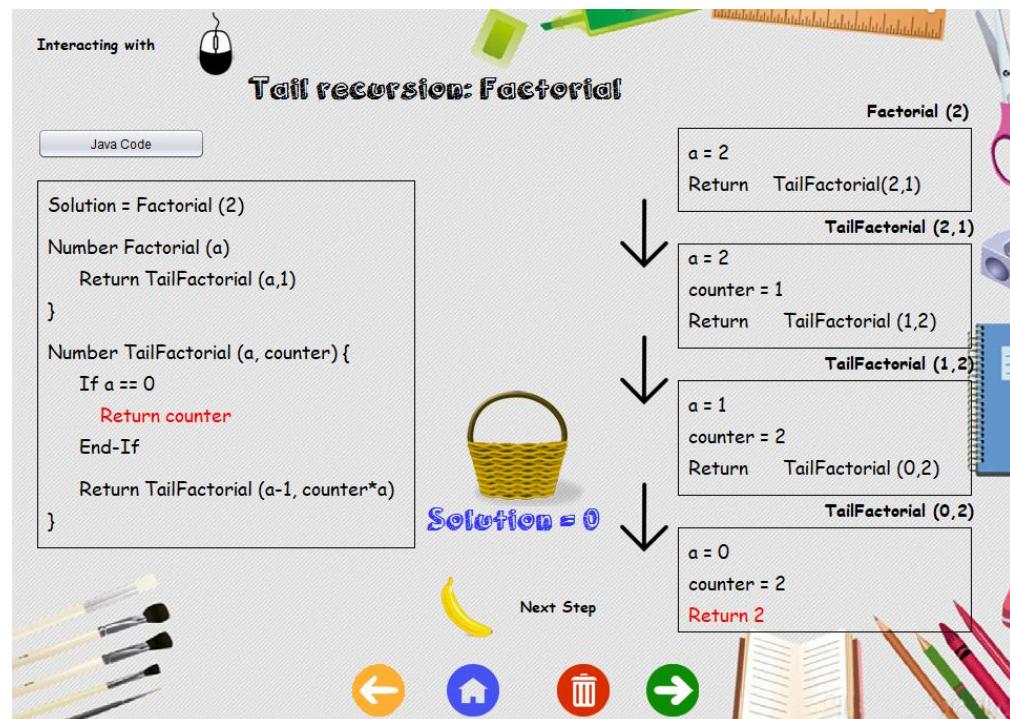


Figura 51. Recursividad por la cola: función factorial.

Fuente: Elaboración propia.

Como se puede observar en la Figura 52, el resultado devuelto en el caso base se pasa a la última llamada recursiva y por tanto esta llamada devuelve el valor 2.

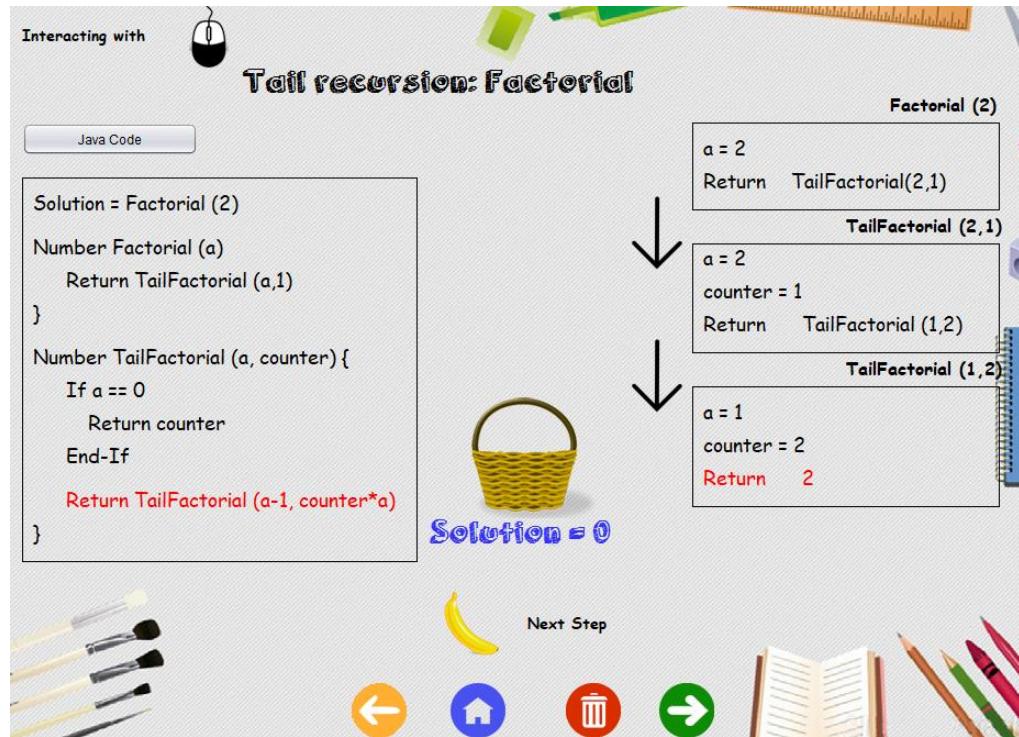


Figura 52. Recursividad por la cola: función factorial. Valor devuelto por la última llamada recursiva.

Fuente: Elaboración propia.

Como estamos en recursividad por la cola, el resto de pasos recursivos o llamadas recursivas se eliminan de memoria y por tanto esta última llamada es la que devuelve el resultado a la función inicial. Finalmente esta función inicial es la que devuelve el resultado. Esto se puede observar en la Figura 53.

Por último el valor devuelto se almacena en la variable solución en memoria, pudiendo observarse en la Figura 54.

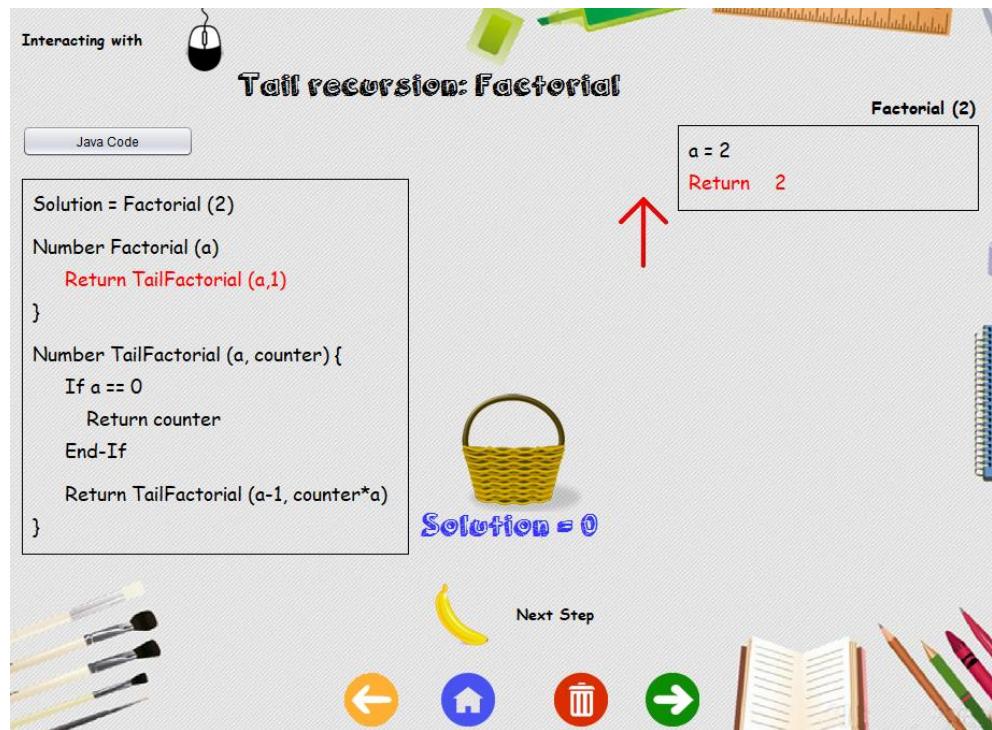


Figura 53. Recursividad por la cola: función factorial. Valor devuelto por la función inicial.

Fuente: Elaboración propia.



Figura 54. Recursividad por la cola: función factorial. Valor almacenado.

Fuente: Elaboración propia.

7.6. Recursividad por la cola: función imprimir suma

En esta última pantalla se pone otro ejemplo de función con recursividad por la cola. Anteriormente habíamos visto la función que se encargaba de sumar dos numeros y devolver el resultado. En este caso nos encontramos con una función que se encarga de sumar dos numeros e imprimir el resultado por pantalla.

Para exemplificar esto, necesitamos una función inicial **Sumar** que se encarga de llamar a la función recursiva **SumarCola**. La función recursiva consta de dos casos base y un caso recursivo.

En la Figura 55 podemos observar a la izquierda el pseudocódigo de las funciones, a la derecha los pasos recursivos hasta llegar a un caso base que es el que imprime por pantalla el resultado.

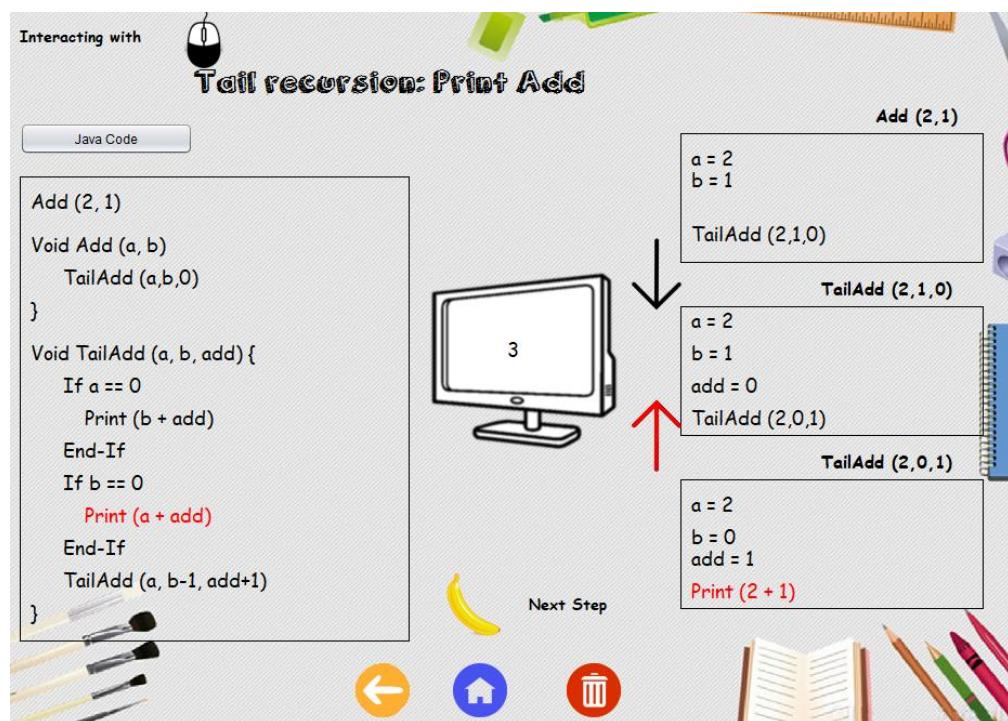


Figura 55. Recursividad por la cola: función imprimir suma.

Fuente: Elaboración propia.

En la Figura 56 observamos que tras imprimir el resultado en el caso base, se vuelve a la última llamada recursiva que es la que eliminará de memoria las llamadas recursivas anteriores.

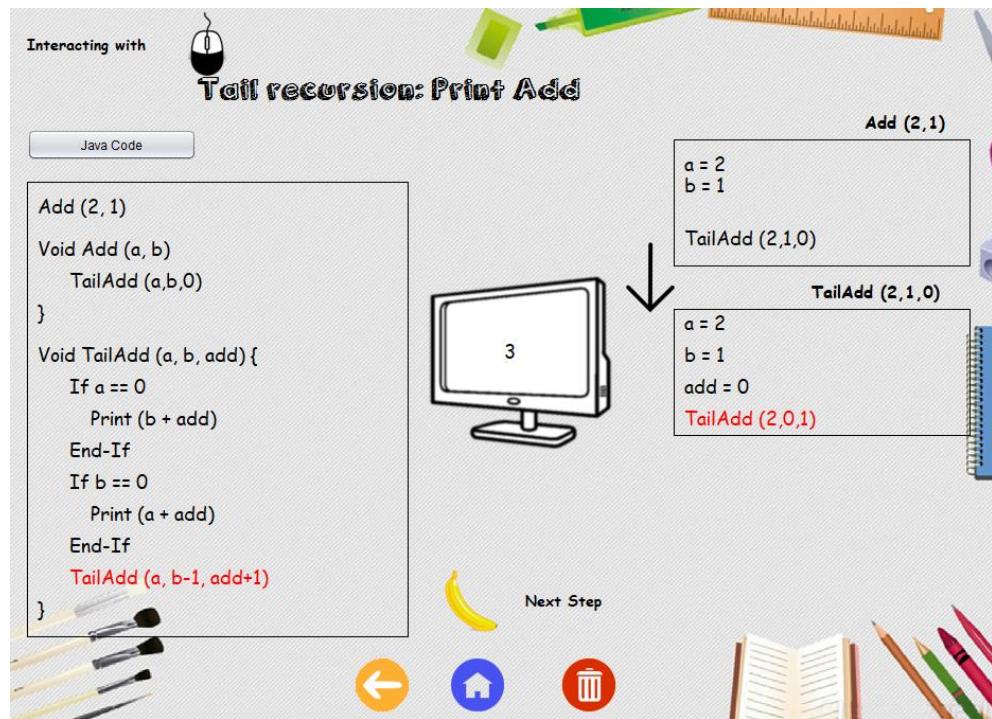


Figura 56. Reecursividad por la cola: función imprimir suma. Última llamada recursiva.

Fuente: Elaboración propia.

8. Web de difusión para la aplicación

Para difundir Primary Code, se ha realizado una página web utilizando la plataforma Google Sites de tal forma que sirva como representación de la aplicación y que sea accesible para todo aquel que esté interesado. En esta página se explica lo que un usuario puede encontrar en la aplicación, los conceptos que puede aprender y además se proporciona un enlace de descarga del programa para ser utilizado en cualquier ordenador.

En esta web un usuario podrá descubrir de qué trata Primary Code y los contenidos que aborda, los modos de interacción que permite la aplicación ya sea mediante ratón o mediante Makey-Makey (vinculando frutas o plastilinas con las teclas), ejemplos a modo representativo de cada categoría o bloque funcional que se ofrece y un enlace de descarga.

La página está disponible en español y en inglés, y se puede encontrar en <https://sites.google.com/view/primarycode-v3/inicio>.

En las Figuras 57, 58, 59, 60 y 61 se puede observar lo que se puede encontrar en la página web.



Figura 57. Web de difusión en Google Sites.

Fuente: Elaboración propia

¿Qué encontrarás en la aplicación?

- Entrada y salida
- Condicionales
- Bucles
- Arrays
- Ficheros
- Funciones
- Recursividad
- ... ¡y una sorpresa!



Figura 58. Web de difusión en Google Sites (categorías)

Fuente: Elaboración propia

¡Novedad! Ficheros

¡Aprende cómo se manejan los ficheros! ¡Practica con ficheros de texto y con ficheros binarios!

Este apartado compara los tipos de ficheros. La sección '¿Qué es un fichero de texto?' explica que son documentos que contienen caracteres y detalles sobre sus flujos de entrada/salida. La sección '¿Qué es un fichero binario?' explica que son documentos que contienen bytes y detalles sobre sus flujos de entrada/salida. Ambas secciones incluyen imágenes de herramientas de oficina y un menú de navegación.

Este apartado muestra un fragmento de código Java para leer de un fichero de texto. El código abre el archivo 'fichero.txt', lee línea por línea y muestra el contenido en la pantalla. Incluye imágenes de un ordenador y un menú de navegación.

Leer datos de un fichero

Aprende a leer datos de un fichero

Figura 59. Web de difusión en Google Sites (ficheros)

Fuente: Elaboración propia

¡Novedad! Funciones

¡Aprende a utilizar los diferentes tipos de funciones!

Una función es un conjunto de líneas de código que sirve para realizar operaciones y devolver un resultado. Las funciones pueden recibir parámetros con los que realizar las operaciones o no recibir nada. Además, no todas las funciones devuelven resultado. Ejemplos de funciones:

- La función **SUMAR** recibe dos números, los suma y devuelve el resultado.
- La función **JUNTAR** recibe dos letras, las junta e imprime las dos letras juntas.

```
Numero Sumar(numero1, numero2){  
    Resultado = numero1 + numero2  
    Devolver Resultado  
}  
Tipos del valor devuelto  
Nombre de la función  
Parámetros de la función  
Valor que se devuelve
```

- La función **SUMARD** recibe dos números, los suma y devuelve el resultado.
- La función **JUNTARD** recibe dos letras, las junta y devuelve una palabra con las dos letras.

Funciones que reciben parámetros v

Figura 60. Web de difusión en Google Sites (funciones)

Fuente: Elaboración propia

¡Novedad! Recursividad

¡Practica con diferentes ejercicios de funciones recursivas! ¡Aprenderás los diferentes tipos de recursividad!

Una función es recursivo si se llama a sí misma, directamente o a través de otra función. En las funciones recursivas hay dos casos:

- Caso recursivo: es el caso más complejo que se va dividiendo en casos más simples.
- Caso base: es la última llamada a la función y se devuelve una solución directamente.

Tipos de recursividad:

- Recursividad lineal: cada llamada recursiva genera una nueva llamada recursiva como mucho.
- Recursividad por la cola: el resultado se devuelve en la última llamada. No se realizan operaciones.

```
Numero Sumar(numero1){  
    Si (numero1 == 1)  
        Devolver 1  
    Fin-Si  
    Devolver Sumar(numero1 - 1) + numero1  
}  
Caso base  
Caso recursivo
```

Solución = Sumar (2, 2)
Número Sumar (a, b) {
 Si (a == 0)
 Devolver b
 Fin-Si
 Si (b == 0)
 Devolver a
 Fin-Si
 Devolver 1 + Sumar (a, b - 1)
}

Sumar (2, 2)
a = 2
b = 2
Devolver 1 + Sumar (2, 1)

Recursividad lineal

Aprende con diferentes ejemplos de recursividad lineal

Figura 61. Web de difusión en Google Sites (recursividad)

Fuente: Elaboración propia

9. Conclusiones

Este proyecto desde sus inicios me entusiasmó mucho dado que me parece muy interesante el tema de la enseñanza. Cuando yo iba al colegio o al instituto no se enseñaba prácticamente nada sobre la informática, lo único que se enseñaba eran conceptos básicos sobre el uso de programas como Word, Excel o Gimp. Me hubiera gustado que en ese momento me enseñasen conceptos relacionados con la programación como los que se enseñan en la aplicación Primary Code. Por este motivo me decanté por esta oferta de Trabajo de Fin de Grado, porque me gustaría que las nuevas generaciones, que desde pequeños ya tienen relación con la informática (redes sociales, videojuegos, etc.), aprendiesen estos conceptos tan importantes de la mejor forma posible para tenerlos enganchados o entretenidos y que disfruten del proceso de aprendizaje.

Además de tener la posibilidad de intentar explicar a los jóvenes mis conocimientos sobre programación de una forma sencilla y muy visual, también he tenido la posibilidad de descubrir herramientas como Makey-Makey, la cual era antes desconocida para mí, cuya utilidad es increíble para hacer que los niños interaccionen de una forma divertida como si se tratase del mando de una consola.

Tras terminar con los desarrollos de las mejoras de la aplicación, he quedado bastante contento con el resultado ya que he seguido con la forma de explicar los conceptos que se usaba previamente y he podido aportar mi granito de arena para que la aplicación sea más completa.

Dicho todo esto, puedo concluir que he cumplido con los siguientes objetivos:

- Se ha rediseñado la pantalla del menú de selección de categoría incorporando los nuevos bloques funcionales (ficheros, funciones y recursividad) y mejorando la estética de la pantalla ordenando los bloques funcionales de forma secuencial.

- Se ha mantenido una estructura similar en los nuevos bloques funcionales para evitar que el usuario perciba que se trata de una versión diferente de la aplicación. Para ello, se han usado cestas de frutas para representar variables en memoria, cajas de bolas de billar para representar arrays, se ha respetado la estructura de las pantallas existentes (bloque de código a la izquierda, explicación y ejemplo visual en el centro, y botones en la parte inferior). También se han introducido nuevas representaciones gráficas para el acceso a un fichero y para las funciones recursivas.
- Se ha explicado de manera visual el concepto de ficheros distinguiendo entre ficheros de texto y ficheros binarios.
- Se ha explicado de manera visual cómo se produce la lectura y la escritura en un fichero (de texto y binario).
- Se han puesto ejercicios de lectura y escritura en ficheros (de texto y binarios).
- Se ha explicado de manera visual el concepto de funciones con representaciones gráficas de las mismas. Se ha explicado cómo son las llamadas, cómo se pasan parámetros a una función y como se devuelve un resultado.
- Se han puesto ejemplos de funciones con parámetros, sin parámetros, con valor devuelto y sin valor devuelto, cambiando el tipo de los parámetros recibidos y devueltos (como parámetros recibidos se han usado enteros y caracteres, como parámetros devueltos se han usado enteros y strings). En el caso de que las funciones no devolviesen un resultado, se ha impreso por pantalla el resultado.
- Se ha explicado qué es la recursividad y se ha diferenciado entre la recursividad lineal y la recursividad por la cola.
- Se han puesto ejercicios visuales sobre recursividad lineal y sobre recursividad por la cola que no sean demasiado complejos. Se ha empezado usando un ejercicio de suma y se ha terminado usando un ejercicio de factorial (incrementando el nivel de complejidad). Se han

puesto funciones recursivas que devuelven resultados y otras que imprimen el resultado por pantalla.

- Para las nuevas funcionalidades mencionadas previamente se ha añadido la funcionalidad de visualizar el código del ejercicio como código Java o como pseudocódigo para que de esta forma el usuario pueda seleccionar la opción más interesante.
- Se ha incorporado funcionalidad para que cada una de las nuevas pantallas añadidas se puedan visualizar tanto en español como en inglés.
- Hacer que las nuevas pantallas agregadas previamente se puedan leer tanto en español como en inglés.
- Se ha creado una web de difusión de la aplicación para poder llegar a los colegios e institutos. En esta web se explica lo que se puede encontrar un usuario en Primary Code V.3. y lo que puede aprender usando dicha aplicación. En esta web se proporciona un enlace de descarga de la aplicación y además se puede visualizar en español o en inglés.

9.1. Dificultades y limitaciones

A lo largo del desarrollo de este Trabajo de Fin de Grado me he encontrado varias dificultades. En primer lugar, me ha sido complicado pensar formas en las que transmitir mis conocimientos sobre ficheros, funciones o recursividad relacionando estos temas con dibujos o imágenes representativas. No es lo mismo intentar explicarle a alguien un concepto que explicárselo a un niño o un joven de tal forma que asocie el concepto a explicar con una imagen y que además le resulte entretenido el tema sobre el que se está hablando.

Además, me ha llevado bastante tiempo adaptar cada ejercicio para que se pudiese visualizar en dos idiomas, el español y el inglés. Al tener botones que cambian el idioma de la aplicación, o que cambian el pseudocódigo por código de Java, requiere mucho tiempo y tener mucho cuidado para que los textos queden correctamente.

Al tener que llevar la aplicación a una web en la que los usuarios puedan descargar el jar de la aplicación para poder ejecutarlo y además explicar en dicha web los conceptos que enseña Primary Code, me ha llevado algo de tiempo ya que nunca había usado Google Sites para crear páginas web, pero al final también es importante utilizar herramientas nuevas para aprender cosas nuevas.

9.2. Mejoras y trabajos futuros

Aunque todos los conceptos incluidos en este TFG para la aplicación PrimaryCode, añadidos a los existentes previamente, abarcan un amplio conocimiento de la programación básica en Java, sería muy interesante que en futuras ampliaciones de la aplicación se introdujesen conceptos nuevos como la programación orientada a objetos, puesto que es algo bastante importante y se utiliza mucho tanto en asignaturas universitarias como en el ámbito laboral.

10. Bibliografía

- Fuentes, J.F.. (2003). *Capítulo 2: Ingeniería de Software, Análisis y Diseño*. Febrero 14, 2021, de Universidad de las Américas Puebla Sitio web: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/capitulo2.pdf
- Bisbal, J. (2009). *Manual de Algorítmica: Recursividad, complejidad y diseño de algoritmos*. Barcelona: Editorial UOC (pp 39-50).
- Vivo, C. (2017). *La competencia digital en edades tempranas: una experiencia educativa con Makey Makey* (Trabajo de Fin de Grado). Castellón: Universitat Jaume I.
- Ceballos, F. (2015). *JAVA. Interfaces gráficas y aplicaciones para Internet. 4^a Edición*. Madrid: RA-MA S.A. Editorial y Publicaciones.
- UNICEF. (2017). *El estado mundial de la infancia 2017. Niños en un mundo digital*. Septiembre 20, 2020, de UNICEF Sitio web: <https://www.unicef.org/media/48611/file>
- Primary Code. (2018). Primary Code V.2. Octubre 25, 2019, de Primary Code Sitio web: <https://sites.google.com/view/primary-code/>
- Ceballos, F. (2006). *Java 2: Lenguaje y Aplicaciones*. Madrid: RA-MA S.A. Editorial y Publicaciones (pp 201-273).
- Universidad Politécnica de Madrid (2013). *Presente y futuro de la Informática*. Revista UPM Universidad politécnica de Madrid, 24, 38 (pp 3-6).